

**PROBABILISTIC FORMAL MODELING AND
VERIFICATION OF REAL TIME TASK SCHEDULING
ON MULTIPROCESSOR SYSTEMS**



Author

FAUZIA EHSAN

Master of Science in Computational Science and Engineering

RESEARCH CENTER FOR MODELING & SIMULATION (RCMS)

NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY

(NUST), ISLAMABAD, PAKISTAN

September, 2017

**PROBABILISTIC FORMAL MODELING AND VERIFICATION
OF REAL TIME TASK SCHEDULING ON MULTIPROCESSOR
SYSTEMS**



By

FAUZIA EHSAN

NUST201463253MRCMS64014F

Supervised by

DR. JAMIL AHMAD

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

in

Computational Science and Engineering

**RESEARCH CENTER FOR MODELING & SIMULATION (RCMS)
NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY
(NUST), ISLAMABAD, PAKISTAN**

September, 2017

Dedicated to

“The Teacher of the Universe”

(Peace Be Upon Him)

With whose existence

and

by having the charity of His knowledge

the cosmos got illuminated with the light of

insight and wisdom

and

the journey of human enlightenment was made possible.

STATEMENT OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Date

FAUZIA EHSAN

Acknowledgments

All the glories, praises and Love are for **ALMIGHTY ALLAH**, the Most Merciful, the Most Gracious, Who bestowed me the thinking power, ability and potential for donating a diminutive scientific knowledge, wet eyes and shuddering lips praise for HIS beloved **The Holy Prophet Muhammad (Peace be Upon Him)** for enlightening my integrity with the quintessence of faith in Allah, congregating all His kindness and mercy upon him.

I would like to take this opportunity to acknowledge my debts to my teachers, friends & family for their kind and generous support and to all wherefrom I have learnt a lot. I do not find words to pay my deepest gratitude to **Dr. Jamil Ahmad**, HoD Computational Science Department, RCMS NUST; the Advisor of this research work, who is not only an authority but unquestionably a visionary personality. I am verily unable to pay his debt for his rational supervision, kindness, extraordinary support and fatherly behavior.

I would also like to say a word of thanks to my GEC members **Engr. Sikandar Hayat Mirza, Dr. Mian Ilyas Ahmad** and **Engr. Fawad Khan** for their valuable guidance and scholarly criticism.

I extend deep emotions of appreciation, expressing my gratitude and indebtedness to **Mr Muhammad Tariq Saeed**, Assistant Professor, RCMS, NUST, **Ms Amnah Siddiqa** (PhD Scholar) and **Mehreen Tahir** (MS Scholar); who have truly influenced the final shape of this work with their invaluable feedback and logistic input.

I truthfully owe my deep gratitude and indebted to a large number of authors and publishers, wherefrom I have benefited a lot in terms of invaluable research material, which has added value to this study.

I am dumb to pay my heartfelt tribute to all those humble beings that have assisted me in any way to become what I am today, whose sacrifices seeded my success, especially **my parents** who have felt my pain beyond me and showered me with never ending prayers and support. I deem them as a divine source of inspiration.

I am also grateful to **Higher Education Commission (HEC), Pakistan** for providing all the financial support.

In the end, I humbly pray to **ALLAH ALMIGHTY** to help and guide us to serve the humanity generally and our great Nation particularly; and to prove us out to be genuinely a valuable asset to our motherland.

Fauzia Ehsan

Contents

List of Abbreviations	v
List of Tables	vi
List of Figures	vii
Abstract	x
1 Introduction	1
1.1 Overview	1
1.2 Real Time Systems	3
1.3 Real Time Scheduling	5
1.4 Problem Statement	8
1.5 Research Aim & Objectives	8
1.6 Our Contributions	9
1.7 Organization of the Thesis	10

2	Literature Review	11
2.1	Petri net Based Modeling for Real-Time Scheduling	11
2.2	Formal Verification and Scheduling	13
2.3	Real Time Scheduling Algorithms	14
2.3.1	Global Earliest Deadline First ($g\mathcal{EDF}$) Scheduling Algorithm .	19
3	Methodology	22
3.1	Methodology Overview	22
3.2	Petri Nets	24
3.2.1	Extended Petri Nets	26
3.3	Non Autonomous Petri Nets	29
3.3.1	Timed Petri Nets	29
3.3.2	Stochastic Petri Nets	30
3.3.2.1	Semantics of Stochastic Petri Nets.	32
3.4	Colored Petri Nets Framework	35
3.4.1	Colored Stochastic Petri Nets	39
3.5	Formal Verification	40
3.5.1	Model Checking	40
3.5.2	Probabilistic Model Checking	41
3.5.3	Continuous Stochastic Logic Model Checking over CTMC . .	44
3.6	Software Environment for Modeling & Verification	47

4	Model Formulation	48
4.1	<i>SPN</i> Model of General Job Scheduler	48
4.2	Colored Stochastic Petri Net of <i>gEDF</i> Scheduling Algorithm	52
4.2.1	CLOCKS MODULE	57
5	Results & Discussions	64
5.1	General Job Scheduling Simulation Results Using Snoopy	65
5.2	Simulation Results of <i>SPN^c</i> Model of <i>gEDF</i> Using Snoopy	68
5.3	Verification Results	79
5.3.1	Stochastic Model Checking of <i>SPN^c</i> Model of <i>gEDF</i>	83
6	Conclusions & Future Work	87
6.1	Conclusions	87
6.2	Future Work	89
	Bibliography	91
	Appendix	106
A	<i>CTMC</i> of <i>SPN</i> model of General Job Scheduling	106

List of Abbreviations

HRTS	<i>hard real time system</i>
SRTS	<i>soft real time system</i>
\mathcal{PN}	<i>standard Petri net</i>
$T_a\mathcal{PN}$	<i>timed Petri net</i>
SPN	<i>stochastic Petri net</i>
CPN	<i>colored Petri net</i>
SPN^c	<i>colored stochastic Petri net</i>
DTMC	<i>discrete-time Markov chain</i>
CTMC	<i>continuous-time Markov chain</i>
CTL	<i>computational tree logic</i>
TCTL	<i>timed computational tree logic</i>
PCTL	<i>probabilistic computational tree logic</i>
CSL	<i>continuous stochastic logic</i>
$gEDF$	<i>global earliest deadline first</i>

List of Tables

1.1	HRTS versus SRTS	5
3.1	Graphical Notations of directed arcs for Extended \mathcal{PN}	28
3.2	Declarations for the \mathcal{CPN} model of Figure 3.5	38
4.1	Description of Places in Figure 4.1	51
4.2	Description of Transitions in Figure 4.1	51
4.3	Declarations for the \mathcal{SPN}^c model of Figure 4.2	54
4.4	Initial Markings defined in \mathcal{SPN}^c model illustrated in Figure 4.2	61
4.5	Description of Set of Places in Figure 4.2	62
4.6	Interpretation of Transitions in Figure 4.2	63
5.1	Rate Functions of transitions in \mathcal{SPN}^c model (Figure 4.2)	70

List of Figures

1.1	Safety Critical Task	4
1.2	Soft-deadline based Task	4
1.3	A Common Real Time Resource Management System	7
2.1	Mono-processor Scheduling	16
2.2	Multiprocessor Scheduling	16
2.3	Flow Chart of global Earliest Deadline First Scheduling Algorithm . . .	20
3.1	Proposed Methodology	23
3.2	An example of Standard Petri Net	27
3.3	Timed Petri net model of Simple Traffic Light Signal	30
3.4	Illustration of Stochastic Petri net	34
3.5	Folding and Unfolding of Colored Petri Nets Example	38
3.6	Model Checking Process	41
3.7	Probabilistic Model Checking Process	42

4.1	SPN Model of Job Scheduler	50
4.2	SPN^c Model of $gEDF$ Scheduling Algorithm	55
4.3	Check Module. Hierarchical structure behind Macro Transition “Check”	56
4.4	CLOCKS Module	58
4.5	Model of Clock for Task_ID=1 and DL=2	59
5.1	Stochastic Simulation of General Job Scheduler SPN model presented in Figure 4.1	66
5.2	Dynamic behavior of significant states in SPN model of General Job Scheduler	67
5.3	Stochastic simulation plot of SPN^c model presented in Figure 4.2 . . .	69
5.4	Behavior of different states of SPN^c with respect to Tasks Completion	71
5.5	Scenario 1. Behavior of stochastic simulations of proposed SPN^c model when available resources are equal to number of tasks.	72
5.6	Scenario 2. SPN^c model simulation results of Waiting queue for varying deadlines with constant number of tasks and processors.	74
5.7	Scenario 3 (a). The box plot obtained from simulations of SPN^c model to determine average throughput	75
5.8	Scenario 3 (b). Stochastic simulation plots of SPN^c model to deter- mine throughput by applying $gEDF$ scheduling policy with varying initial multiprocessors	76

5.9	Scenario 4. The stochastic simulation behavior of latency for varying number of processors with constant number of tasks applied to the \mathcal{SPN}^c model of $g\mathcal{EDF}$	78
5.10	Scenario 5. The stochastic simulation results obtained from simulations of \mathcal{SPN}^c model with different initial idle multiprocessors to determine their effect on execution of tasks (RUN state in model) by applying $g\mathcal{EDF}$ scheduling policy	78

Abstract

The emerging multiprocessors technology is increasingly penetrating into the advanced real-time systems which poses great challenges for job scheduling in real-time resource management systems. An efficient and reliable scheduling analysis technique is vital for non-deterministic tasks in safety-critical systems. This study explores state-of-the-art automated stochastic formal techniques for real time scheduling analysis and verification. This is the first work so far that employs colored stochastic Petri nets (\mathcal{SPN}^c) and probabilistic model verification based framework for multiprocessor real-time scheduling algorithms. Stochastic Petri net (\mathcal{SPN}) model of general job scheduler is evaluated via simulations, structural analysis and stochastic model checking. Specifically, this study demonstrates \mathcal{SPN}^c model of preemptive and dynamic priority based global earliest deadline first ($g\mathcal{EDF}$), a non-trivial scheduling algorithm for real time multiprocessor environment. The underlying \mathcal{SPN} model of \mathcal{SPN}^c and its continuous-time Markov chain are deeply examined and validated using stochastic simulations and model checking technique. Hence, this work suggests a framework for the formal analysis of real time scheduling algorithms and it successfully proves to provide a live, scalable and robust solution for $g\mathcal{EDF}$ scheduling algorithm. This study will further serve as a basis to systematically analyze the real time multiprocessor scheduling problems tremendously using probabilistic formal methods.

Chapter 1

Introduction

In this chapter, a brief overview of the research problem is discussed along with the introduction of important terms and concepts related to this research work. The problem statement is given in Section 1.4. The research aim and objectives are stated in Section 1.5 along with our contributions in Section 1.6. A brief outline of the thesis document is given in Section 1.7.

1.1 Overview

Modern safety-critical systems are moving towards increasingly parallel and distributed computing environments. They highly rely on their integrated concurrent systems that are by nature very complex e.g. Tele-presence systems, Automatic tracking systems, E-health system etc.. An efficient modeling of multiprocessor systems for sophisticated

real time applications is particularly a very challenging problem because a single miss in some important interactions will often lead to malfunctioning of whole system and which may cause in ecological, human or economic disaster. The early failure analysis of real time tasks scheduling is very important to protect the system from serious accidents. All the real-time tasks must fulfill their timing constraints and thus real-time job scheduler must be very reliable during development cycle from design to final implementation phase. Over the years, real time scheduling analysis is done using calculus based techniques. In the recent research, mathematics based formal method approaches are widely used that guarantee the reliability of safety critical systems at high abstraction level [1, 2].

The automated stochastic formal modeling and verification techniques are proved to be very useful for predictability and performance properties of the non-deterministic complex systems architectures [3]. The expressive power of Petri nets modeling formalism can be used for performance evaluation of real time scheduling problems with less complexity. Qualitative and quantitative properties verification can be done through model checking which further provides deep insights into the behaviors of modeled system and also precisely validates the correctness of a system. The selection of an appropriate formal method depends on underlying system and its important features to be verified. Therefore, in this study, as opposed to traditional analytical methods, stochastic formal modeling and verification techniques are exploited to investigate the non-deterministic behavior of real time tasks scheduling.

1.2 Real Time Systems

Real-time systems are those reactive computing systems (reactive systems interact with their environment continuously and generates an output on the basis of certain inputs) in which the correctness of the system depends not only on the logical computational results but also on the time instance at which the computational results are produced [4]. The pervasive real time systems are used in many different application areas, for example, Command Control Systems, Air Traffic Control Systems, defence and space systems, networked multimedia systems etc., in order to monitor and control the environment.

If the real time system follows its time constraints stringently then it is known as hard real time system (*HRTS*). If the deadline timing constraints are tolerable then the real time system is said to be soft real time system (*SRTS*). *HRTS* is also known as *safety-critical* because a single miss to meet the deadline time will lead the system to serious damages as shown in Figure 1.1. However, *SRTS* is best-effort depicting that the deadline failures are tolerable but not desired because the system performance degrades as shown in Figure 1.2. Some important differences between *HRTS* and *SRTS* are listed in Table 1.1 [5].

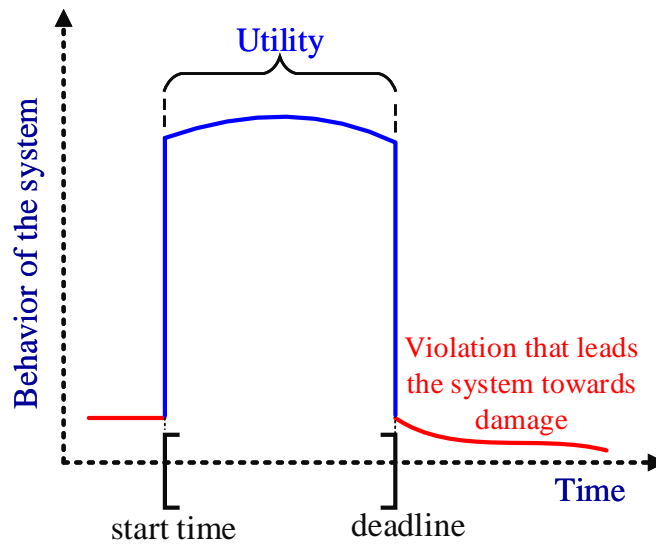


Figure 1.1: Safety Critical Task.

The behavior of figure depicts that if the safety critical task does not complete within deadline then the system will headed towards damage.

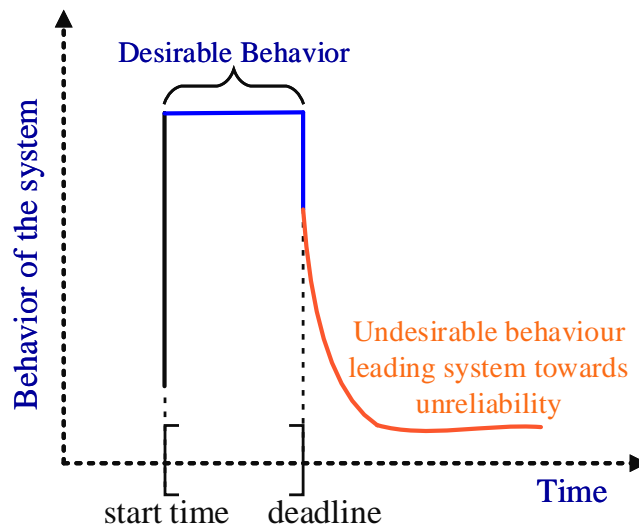


Figure 1.2: Soft Deadline Task.

The behavior of figure depicts that the system will not go to damage state as the task does not complete within deadline but the performance of system degrades.

Table 1.1: Differences between HRTS and SRTS

Characteristic	Hard Real Time	Soft Real Time
Response Time	Hard-required response time that should be in milliseconds or less	Soft-desired response time that are flexible
Peak-load Performance	It must be predictable such that it should not violate the deadlines	Degraded performance peak load are tolerable.
Control of pace	<i>HRTS</i> must remain synchronous with the environment in all the circumstance	Computer dependent i.e. the response time of <i>SRTS</i> slows down in case of high load on computing nodes.
Safety	Critical	Non-Critical
Data Integrity	It has short-term data integrity as it deals with real-time data files	Data integrity in <i>SRTS</i> is long-term as it has to deal with larger databases, e.g. on-line reservation systems
Error Detection	Error detection in <i>HRTS</i> is autonomous and roll-back/recovery of computations is of limited use.	It is user-assisted i.e. the computation is rolled back to a previous checkpoint in order to initialize a recovery action.

1.3 Real Time Scheduling

Real time scheduling theory is the analysis and verification of scheduler system as well as scheduling policy adopted for real-time applications. There came a paradigm shift in real time systems from uniprocessors to multiprocessor computing architectures that brought significant and challenging problems to real time scheduling theory. More efficient and reliable advance scheduler schemes are required to handle parallel jobs in multiprocessor real time systems. The scheduling policy in *HRTS* must provide guaranteed response time [6] as it follows strict timing constraints. On the other hand, the aim of scheduling in *SRTS* is to optimize the average response time (Best effort

scheduling policy) [7] since timing constraints are flexible. In this study, our focus is on real time tasks that have hard deadline constraints.

In HRTS (best-known as safety critical systems), the tasks are executed by multiple processors in such a manner that task strictly completes within its deadline, or else the system will go into catastrophic state which indicates that it needs extreme reliability and safety. A real time resource management system handles the load on processors for all the specified compute nodes by keeping the jobs from competing with each other. A common resource management system is illustrated in Figure 1.3. Job scheduler, a fundamental element in real time resource management systems, is responsible to make scheduling decisions in order to manage all the load on computing nodes justifiably and efficiently [9, 8]. The model of scheduler is not only required but its verification is also necessary. A sound scheduler that cannot verify the truthness of generated schedule is counterproductive in real-time applications because real time system designer needs an efficient scheduling algorithm in which fundamental property of HRTS must be proved (i-e deadline of any task should not be missed) [10]. Thus, soundness of existing schedulability tests plays a central role in the selection of any job scheduler. A stochastic Petri net model of general job scheduler is presented below in Figure 4.1 and is further elaborated in section 5.

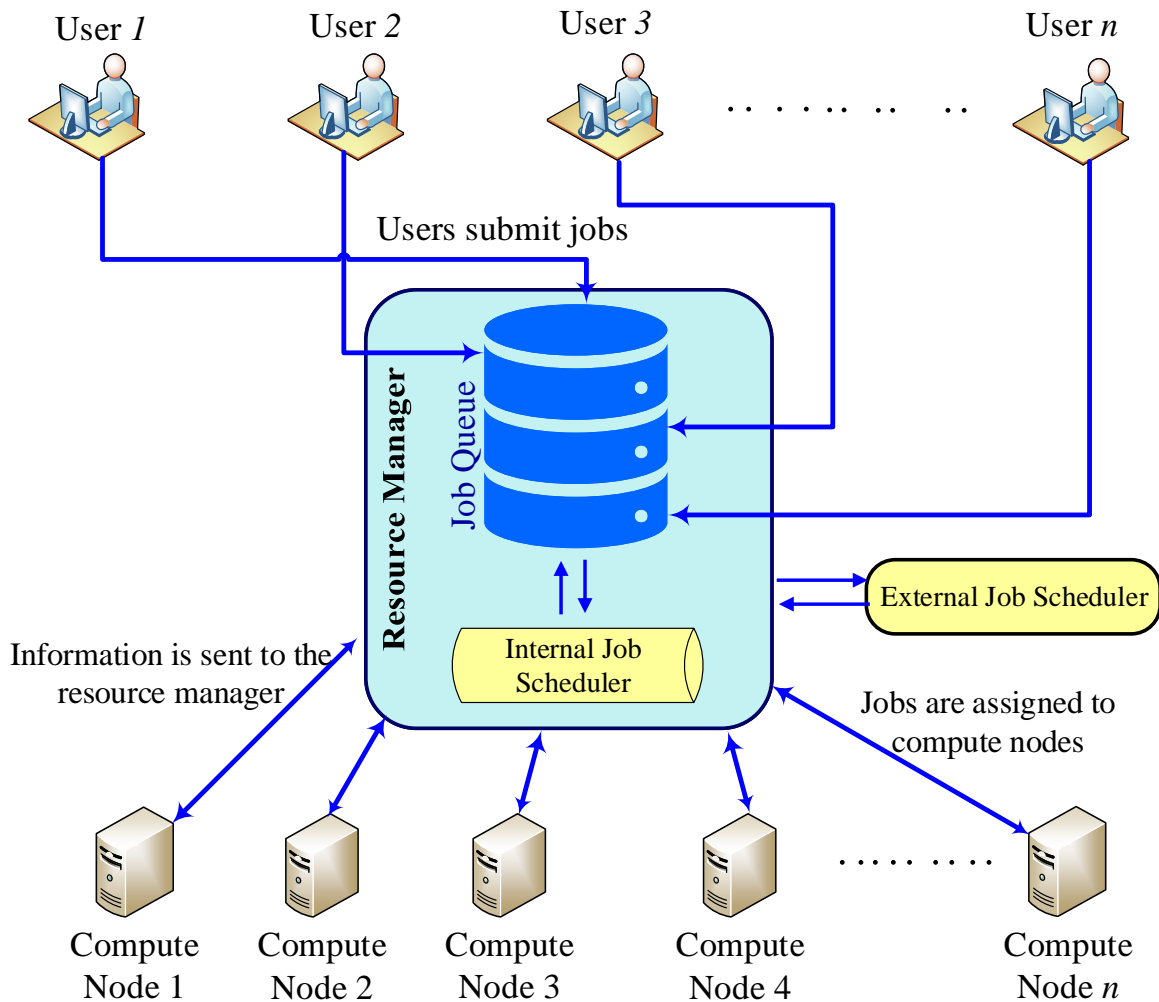


Figure 1.3: A Common Real Time Resource Management System.

Resource management system generally consists of a resource manager and a job scheduler. It has a built-in internal job scheduler but external job scheduler may also be added to enhance the functionality. A resource manager keep track of available compute resources and notify the user about the status of previously submitted jobs. It also assists to organize already submitted jobs on the basis of priority, requested resources and processors availability. The resource manager sends information about job queues and idle resources to the scheduler continuously and makes the schedule keeping into account all the constraints e.g. priority, on the basis of which job scheduler determines the order of jobs execution along with the corresponding compute nodes [8].

1.4 Problem Statement

With the increase in the development of multiprocessors technology, real time systems are becoming complex day by day. These concurrent systems demand a well-defined reliable method in order to manage the computationally intensive and safety-critical tasks efficiently. To investigate the probabilistic nature of job scheduling in real time resource management system, the existing analysis techniques are limited in their scope and generally evaluate the deterministic behavior of real time system. The expressive graphical modeling based colored stochastic Petri net techniques are employed to evaluate the non-deterministic performance of real time scheduling algorithm via multiple simulations. Moreover, probabilistic model checking approach completely verifies the scheduling properties of the stochastic model. Thus, automated stochastic formal methods are proposed to get robust and precise solution for the probabilistic and complex real time job scheduling. They also provide deep insights into the dynamics of the model and completely validate the system.

1.5 Research Aim & Objectives

The aim of this research work is to provide an efficient model based scheduling solution for real time resource management system. This aim is achieved with the following objectives:

- To construct a formal model of colored stochastic Petri nets of real time scheduling algorithms.

- To evaluate the performance of real time scheduling algorithms for multiprocessors via simulations of colored stochastic Petri net (\mathcal{SPN}^c) model.
- Formal verification and validation of continuous-time Markov chain ($CTMC$) through probabilistic model checking approach using PRISM tool.

It will ultimately assist the researchers to combine the efforts of performance engineers and verification engineers by handling modeling and analysis challenges together in a very precise manner.

1.6 Our Contributions

Probabilistic formal modeling and analysis based approach is proposed for real time task scheduling on multiprocessors. Colored stochastic Petri net (\mathcal{SPN}^c) based framework is used for performance evaluation of real time scheduling algorithms. \mathcal{SPN}^c model of Preemptive global earliest deadline first ($g\mathcal{EDF}$) algorithm is constructed in Snoopy tool and the dynamic behavior of $g\mathcal{EDF}$ is analyzed via multiple simulations. Furthermore, a probabilistic model checker PRISM is used to build continuous-time Markov chain ($CTMC$) which can assist to formally validate and verify the model. A simple stochastic Petri net (\mathcal{SPN}) model of General Job scheduler, one of the critical element in resource management system, is also formulated. Structural properties of model are analyzed through the Charlie tool. The framework comprises of colored Petri nets and the probabilistic model checking plays a dominant role in evaluating the per-

formance of real time job scheduling problem on multiprocessors. Thus, our approach permits precise analysis of random behavior of real time systems. This study will play a significant role to further explore the problems in multiprocessor real time scheduling area.

1.7 Organization of the Thesis

The thesis document is organized in the following manner. In Chapter 1, brief introduction of the problem is stated along with the discussion of some important terms and concepts that are related to real time systems and its scheduling. Chapter 2 provides the extensive review of formal modeling and verification techniques for real time scheduling. In addition to that, real time scheduling algorithms are also discussed. The techniques adopted for modeling and verification are demonstrated in Chapter 3 along with the proposed methodology. SPN model of general job scheduler and SPN^c model of $gEDF$ are formulated in Chapter 4. The results obtained via simulations, quantitative & qualitative analysis findings are elaborated and discussed in Chapter 5. Subsequently, the study is concluded on the basis of results in Chapter 6. Finally, the potential future directions using the proposed methodology are given in Section 6.2.

Chapter 2

Literature Review

This chapter provides an overview of research background on formal modeling and analysis based approaches to workout real time scheduling problem. Section 2.1 elaborates modeling based techniques that were applied to solve real time scheduling problem with the main focus on Petri nets (\mathcal{PN}). The literature of formal verification based approaches for real time scheduling is briefly explained in section 2.2. Section 2.3 discusses the real time scheduling algorithms and further demonstrates the reason for selection of global earliest deadline first ($g\mathcal{EDF}$) scheduling algorithm in section 2.3.1.

2.1 Petri net Based Modeling for Real-Time Scheduling

Over the past two decades, real time scheduling problem has been modeled by utilizing various approaches e.g. timed automata (**TA**) [11, 12], high order logics (**HOL**)

theorem proving method [13], Petri nets ($\mathcal{PN}s$) [14] etc., in order to obtain its better understanding and solutions. The models based on Petri nets and its extensions provide immense articulation, dynamic behavior and simulation features, that make it favourable for modeling and analysis of real time scheduling problem. $\mathcal{PN}s$ simulations make it possible to detect system design errors and violation of time constraints in early design phases prior to the deployment of real time systems. Martin Naedele suggested a novel approach of using Petri nets for modeling of real time systems and develop general model for scheduling of tasks on a uniprocessor [15]. W. M. P. van der Aalst mapped scheduling problem to timed Petri nets for the first time to model and analyze the scheduling problem. Few structural properties and behavioral properties are discussed in this study e.g. reachability graphs are used to determine feasible schedules. This link between scheduling and Petri nets stimulated further research in scheduling and Petri net analysis [16]. S. BALAJI et al. introduced S-nets for performance evaluation of real time scheduling algorithms using scheduler block along with deterministic timed Petri nets [17]. Koriem, S. M used real-nets (R-nets) for performance analysis of HRTS using extended version of time Petri nets with probability density function [18]. In [19], periodic real time tasks of off-line scheduling algorithms are analyzed using state graph of Petri net models. SETPN (Scheduling Extended T-time PNs) are proposed in [20] to analyze the static scheduling policy using extended t-timed PNs [21]. Priority extension of time Petri nets known as dynamic priority time Petri nets (dPTPN) are introduced for partitioned multiprocessor schedulability verification of periodic task

sets of real time systems [22]. The literature shows that Petri net and its extensions used for real time scheduling are becoming more complex for large parallel systems and it is difficult to analyze non-deterministic behavior of real time scheduling. Hence, the current work proposes scalable, parameterized and powerful colored stochastic Petri net (SPN^c) based approach for real time scheduling algorithms. To evaluate the performance of the system, a number of simulations are carried out for the verification purposes.

2.2 Formal Verification and Scheduling

Formal verification makes it possible to explore all the potential and achievable system behaviors [23]. In this way, it not only provides performance estimates but also guarantees system's performance. Most of the previous studies are based on temporal logic reasoning [24, 25, 26]. Model checking, an automated and rigorous formal verification approach is not widely exploited to workout real time scheduling problem. In [27], the authors proposed model checking approach for static scheduling of periodic real time tasks as an alternative to quantitative logic reasoning. Ranjeev et al. [28] applied model checking approach over DTMC with time based branching temporal specifications *TCTL* for real time systems. Real time systems are formally verified under preemptive scheduling policy in [29]. Some tools are also established for scheduling of real time applications. For example, *Roméo* that performs on-the-fly *TCTL* model-

checking of Petri nets. *Cheddar* tool is based on Ada framework for temporal behavior analysis of real time scheduler but it does not perform model checking [30]. *Kronos* tool is introduced for analysis of bounded response and reachability properties of real time systems using timed automata and *TCTL* [31]. A model checker tool *PARC* for verification of dPTPN in scheduling of real time systems is proposed by Karamti and Mahfoudhi [32]. Recently, Liu et al. [33] formally model and verify Rate Monotonic Scheduling (RMS) algorithm using logical reasoning in real-time Maude formal modeling tool.

Formal model checking approaches applied to real time scheduling are not well established and is employed in very few works even in emerging real time applications domain. It is still designed at low abstraction level using empirical techniques, that could not guarantee the reliability and predictability of real time applications, consequently, system may collapse [34]. Probabilistic formal model checking approaches are proved techniques that ensure the reliability in real time applications [35]. Therefore, probabilistic model verification based approaches are used that make use of continuous-time Markov chain (CTMC) which is not utilized till now, to the best of our knowledge, in multiprocessor real time scheduling field.

2.3 Real Time Scheduling Algorithms

Real time Scheduling Algorithm defines a rule set to manage real time tasks/programs in strict and timely manner. Although extensive work has been done for scheduling of

real time multiprocessor platforms since 1969 but real time community is still working hard to find out effective and optimal scheduling algorithms for multiprocessor computing systems [36, 37]. These systems have their own intricacies; consequently, uniprocessor scheduling algorithms cannot be applied directly to multiprocessor systems [38]. Real time multiprocessor systems are developing day by day with multi-core and many-core processors technology and are widely exploited in computationally intensive applications [39]. As the real time systems are those reactive systems that not only need logical correctness but temporal verification is also essential. Formal analysis techniques not only boost the guaranteed performance in real time applications but also provide accurate proofs of schedulability. Therefore, it is vital to use more expressive and significant formal methods in order to fully utilize the computing resources such that all the jobs are completed within their deadlines [40].

Uniprocessor scheduling is a 1-dimensional problem that only deals with the temporal organization of tasks as depicted in Figure 2.1. As opposed to uniprocessors, multiprocessor scheduling is a 2-dimensional problem as presented in Figure 2.2, i.e. in addition to the temporal management of tasks, it also deals with their spatial organization [7]. For every job set, it determines when the task should begin, suspend and/or terminate (*priority problem*) as well as identify the processor to which task should be assigned (*allocation or migration problem*) [36].

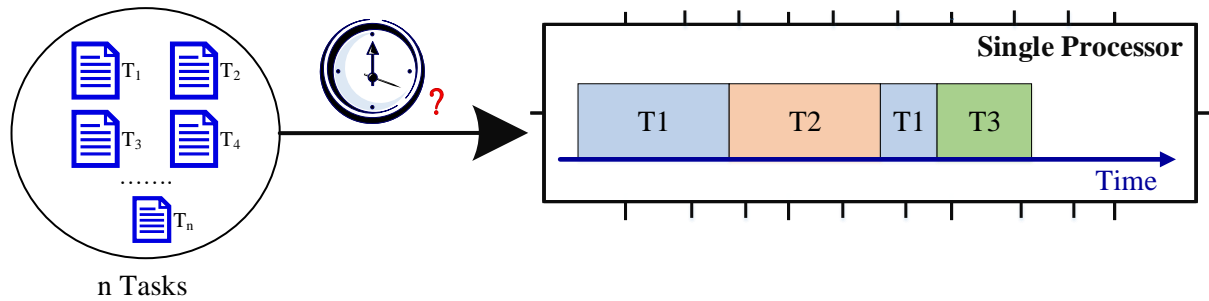


Figure 2.1: Mono-processor Scheduling.

A uniprocessor scheduling is one dimensional problem that takes care of only the temporal management of all the tasks i.e. the time at which the task should be started, preempted and/or resumed.

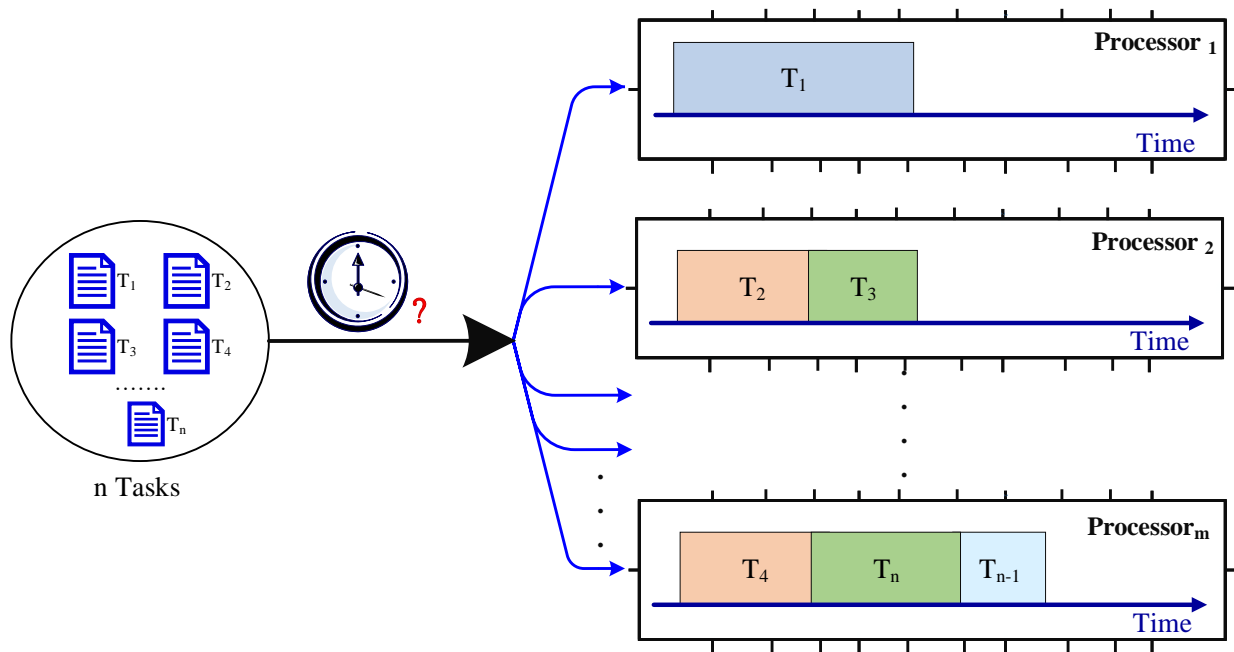


Figure 2.2: Multiprocessor Scheduling.

A multiprocessor scheduling is two dimensional problem that deals with both the temporal as well as spatial organization of tasks i.e when and to which processor the tasks should be assigned

On the basis of migration problem, scheduling algorithms for multiprocessors are classified into two groups:

- (a) **Global scheduling algorithms:** A task can be run and migrated to any processor such that there is no pre-allocation of tasks to processors
- (b) **Partitioned scheduling algorithm:** There is no migration of tasks allowed once it is assigned to the processor.

On the basis of priority problem the scheduling algorithms are classified as dynamic-priority and static-priority scheduling algorithms [41, 42]. In the former approach, the active jobs are being checked for priorities at each time instant. These type of schedulers are also known as on-line schedulers because the scheduling decisions are taken at execution/run time of real-time application [43]. In static priority scheduling algorithm, the jobs priorities are fixed and all the scheduling decisions must be taken a priori. It is also known as fixed priority clock-driven scheduling [44]. Static scheduling algorithms are quite limited and uncompromisable to be used in highly non-deterministic real time systems whereas dynamic algorithms are flexible enough to change scheduling decisions at run time based on overloads occurrence e.g. burst of task arrivals, malfunctioning of few machine parts etc.. Thus, dynamic scheduling algorithms are significantly used to manage the random occurrences of real time tasks [45]. If the scheduling policy allows the jobs to be interrupted i.e. suspended temporarily and then resumes later to another processor then scheduling algorithm is said to be preemptive.

In contrary to that a scheduling algorithm is non-preemptive if the job, once in running phase, could not be terminated till the completion of task [36]. This research work considers global, preemptive and dynamic priority based real time scheduling algorithms. Different multiprocessor scheduling algorithms have been proposed to achieve user as well as high system level performance and reliability. As in real time systems, in addition to predictability and fairness that are essential factors, efficiency is also important [46, 47]. Therefore, the most desirable characteristics of hard real-time scheduling algorithms are:

- It must guarantee and ensure average response time. The optimization of response time is the secondary goal.
- It must guarantee that the deadlines are always met if the tasks have hard real time constraints.
- The degree of resource utilization should be high or maximum. The miss ratio must be zero.
- The scheduling stability is guaranteed even during temporary overloads i.e. deadlines of critical tasks must be met (task should not expire) even during overload conditions. Such scheduling algorithm is said to be stable.

These are the important objectives on the basis of which scheduling algorithm is evaluated and its rank is determined for real-time applications.

In this study global earliest deadline first ($g\mathcal{EDF}$), a well known real time multiprocessor scheduling algorithm, is selected which provides a basis for analysis and verification of dynamic behavior of real time scheduling problem. $g\mathcal{EDF}$ algorithm is discussed in section 2.3.1.

2.3.1 Global Earliest Deadline First ($g\mathcal{EDF}$) Scheduling Algorithm

$g\mathcal{EDF}$ is a deadline driven preemptive scheduling algorithm for multiprocessor real time systems [45]. The task is prioritized on the basis of how much a task is close to its deadline. A flow diagram of $g\mathcal{EDF}$ is illustrated in Figure 2.3. $g\mathcal{EDF}$ is a deadline based dynamic priority scheduling algorithm in which jobs with the least deadline has given the highest priority.

Let $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ be the set of sporadic tasks then the tasks are processed by \mathcal{EDF} algorithm such that :

$$Priority(\tau_i) > Priority(\tau_{i+1}) \iff DL(\tau_i) < DL(\tau_{i+1}) \forall i \in \{1, 2, \dots, n\}$$

where ' DL ' is the absolute deadline.

\mathcal{EDF} when used for real time multiprocessor applications is known as global \mathcal{EDF} . As it is global scheduling algorithm so jobs are assigned dynamically from ready queue to computing nodes. There is a single queue for all the processors which gives better response on average instead of one queue per processor as in case of partitioned scheduling [48]. $g\mathcal{EDF}$ is a part of planning based real time scheduling algorithms

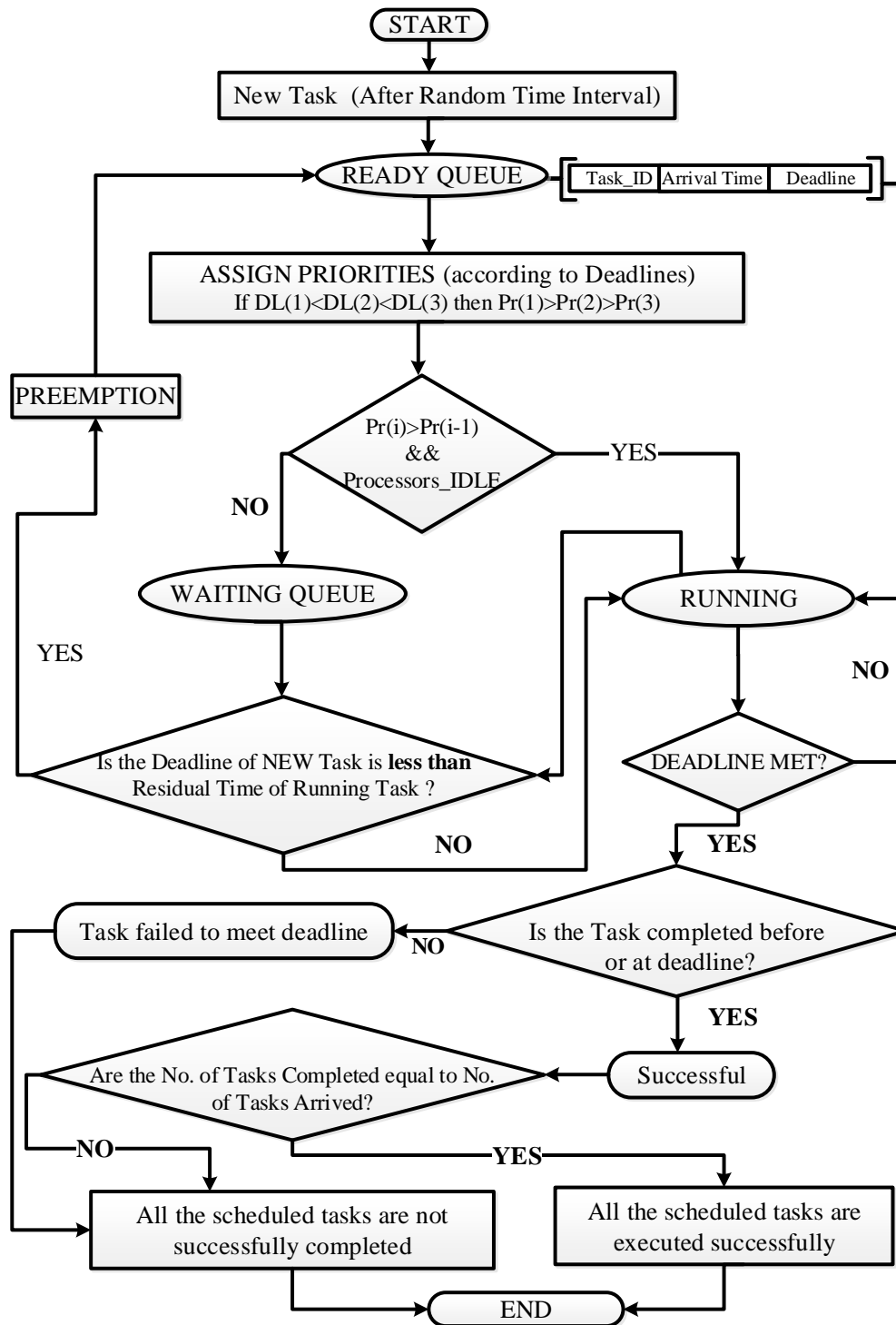


Figure 2.3: Flow Chart of Global Earliest Deadline First Scheduling Algorithm

because scheduling decisions are taken such that all the tasks meet their deadlines [49]. The scheduling analysis in multiprocessor systems must be specified using dynamic priority [50]. Moreover, $g\mathcal{EDF}$ is a memoryless algorithm since it takes scheduling decisions only over current state of the system and active/ready tasks at the present time 't', irrespective of previous scheduling decisions [36]. The motivation behind selection of $g\mathcal{EDF}$ scheduling algorithm is its memorylessness and dynamic nature which is coherent with stochastic formal techniques since they also exhibit memoryless property.

Chapter 3

Methodology

3.1 Methodology Overview

The scheduling theory of real time systems is a framework which further provides analytical methods, called **schedulability tests**. It allows system designer to early analyze/compute the system's behavior before implementation/execution phase. The modeling techniques provide a basis for systematic designing of real time systems and extremely simplify their verification [51]. Building a model for scheduling of modern safety critical systems is a non-trivial problem. Job scheduler plays a functional role in the resource management of real time systems as discussed in Introduction (Section 1) which indicates that scheduler modeling is a very challenging problem.

Traditional scheduling analysis techniques are restricted in their scope to quite simple systems. Moreover, existing implementation theories lack verification of functional model level properties such as liveness, deadlock freeness and fairness properties. The

proposed methodological framework for modeling real time scheduling is presented in Figure 3.1 and explained below:

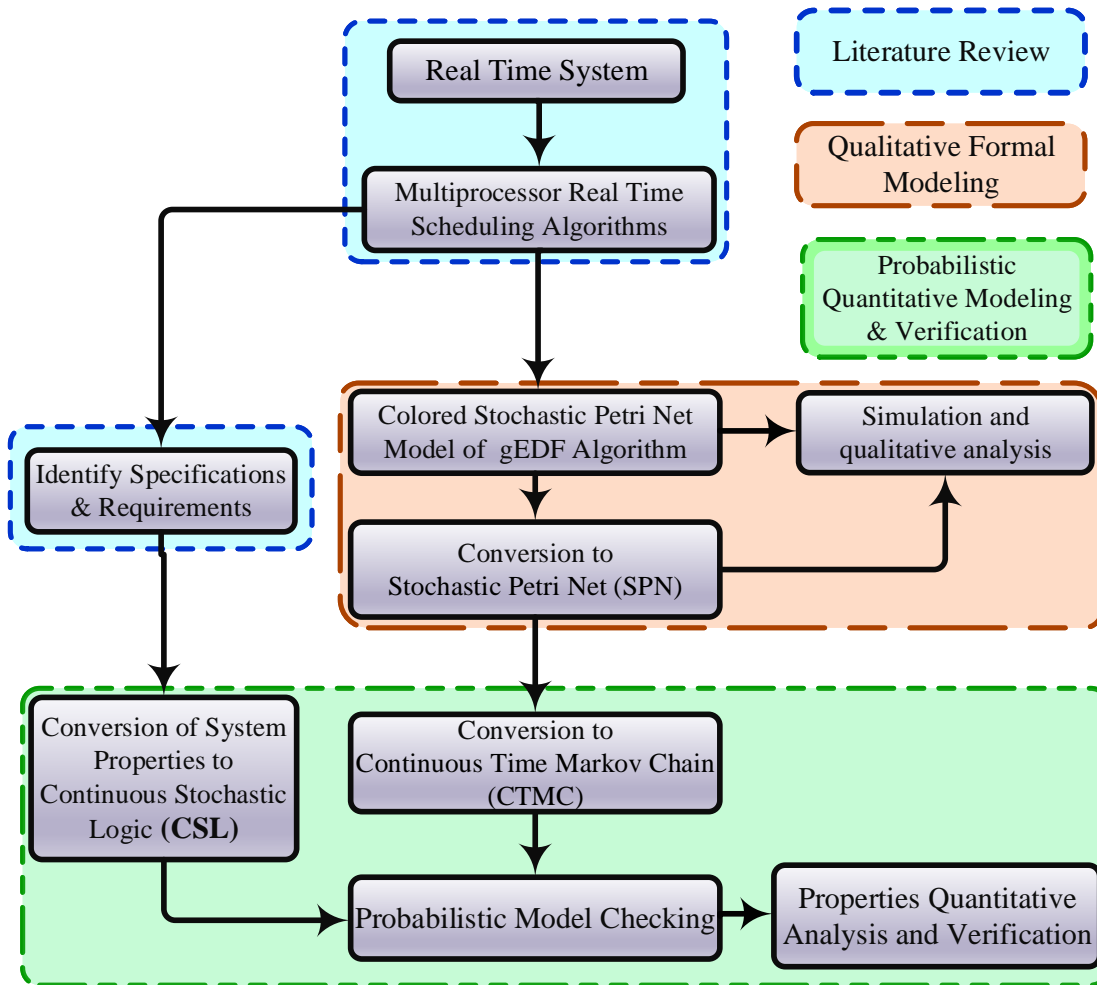


Figure 3.1: Proposed Methodology.

An overview of the organization and structure of the study is presented in this workflow diagram. After the comprehensive literature research of real time systems and their scheduling algorithms, global earliest deadline first ($gEDF$) scheduling algorithm is selected. It is modeled using colored stochastic Petri net in *Snoopy* tool. The model is analyzed through simulations and qualitative analysis by using *Snoopy* and *Charlie* tool respectively. The formal model properties of the system are further analyzed quantitatively through probabilistic model checking via *PRISM* tool that takes (i) *CTMC of probabilistic model* and (ii) *CSL encoded system specifications*, to completely verify the system.

3.2 Petri Nets

Petri net is a very powerful mathematical and graphical modeling tool used to visualize the behavior of concurrent, asynchronous, deterministic and/or non-deterministic systems. In 1962 Carl Adam while doing his PhD thesis “**Kommunikation mit Automaten**” [Communication with automata] gives the idea of Petri nets which significantly advanced the fields of distributed, parallel computing. In 1989 TADA0 MURATA [52] wrote the first review paper of Petri nets. $\mathcal{PN}s$ are very suitable for concurrent, parallel and distributed discrete event dynamic systems. Now after more than half a century, $\mathcal{PN}s$ are extensively used to solve multidisciplinary real world problems. The diverse applications of this mathematical and logical formalism are still an ongoing research area [53].

\mathcal{PN} is basically a directed bipartite graph (see Definition 1 & 2) in which directed arcs are used to connect places and transitions. The places usually represent state variables whereas transitions are “state transformers” usually express the actions/events to neighboring places. The tokens allocated to all places are termed as markings. A place ‘ p ’ is an input place if there is a directed arc from ‘ p ’ to transition ‘ t ’ and if there is a directed arc from ‘ t ’ to ‘ p ’ then ‘ p ’ is an output place. A transition is **enabled** or **fire-able** if each of the input place contains tokens equal to the weight of the transition. The firing of enabled transition extract tokens from input place(s) and adds tokens to output place(s) equal to weight of the arc joining transition to place [54]. These simple rules of

transition firing results in the trajectories of marking. In this way, the dynamic behavior of the system is produced. Formally, standard Petri net is explained in Definition 2.

Definition 1 (Directed Graph) A directed graph G is a pair $\langle A, \xi \rangle$ in which:

- A is a set of all nodes / vertices $\{v_1, v_2, \dots, v_n\}$, and
- ξ is ordered pair of edges such that if (e_1, e_2) is an arc from e_1 to e_2 then it is distinct from (e_2, e_1) i.e. $(e_1, e_2) \neq (e_2, e_1)$. Edges are used to connect vertices.

Definition 2 (Directed Bipartite Graph) A directed graph $G (A, \xi)$ is bipartite in graph theory, iff:

- $A = \mathcal{P} \cup \mathcal{T}$ such that $\mathcal{P} \cap \mathcal{T} = \emptyset$ where \mathcal{P} and \mathcal{T} are two mutual exclusive sets of nodes, and
- $\xi \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a directed arc from \mathcal{P} to \mathcal{T} or from \mathcal{T} to \mathcal{P} .

For example, in Job Assignment problem the resources and jobs are two disjoint sets of vertices and edges. These are used to represent the compatibility of Jobs with different available resources.

Definition 3 (Standard Petri Net) A standard Petri net is a quadruple $\mathcal{PN} = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}_w, I_0 \rangle$, where :

- \mathcal{P} is a bounded, non empty set of places $\{p_1, p_2, \dots, p_n\}$
- \mathcal{T} is a bounded, non empty set of transitions $\{t_1, t_2, \dots, t_m\}$

- $\mathcal{F}_w : ((T \times P) \cup (P \times T)) \rightarrow \mathbb{Z}_{\geq 0}$ is a weight function assigned to directed arcs. It is basically an arc multiplicity that maps a positive integer to ordered pair of edges (p_i, t_j) and (t_j, p_i) , $\forall p_i \in \mathcal{P}, t_j \in \mathcal{T} \ \& \ i, j \in \mathbb{N}$
- $I_0 : \mathcal{P} \rightarrow \mathbb{Z}_{\geq 0}$, is function of the initial marking which assigns discrete positive marks to set of places and leads to further markings, thus exhibits the dynamic portion of Petri net model.
- $\mathcal{P} \cap \mathcal{T} = \varphi \ \& \ \mathcal{P} \cup \mathcal{T} \neq \varphi$ i.e. places and transitions must be two mutual exclusive partites. The example of simple Petri net model of machine cycle in Fig3.2 illustrates the semantics of standard Petri net \mathcal{PN}

3.2.1 Extended Petri Nets

Petri nets can be extended by adding some special features of directed arcs, for instance, read arcs and/or inhibitor arcs. These two additions simplify the model to represent positive and negative actions/conditions comparatively in an easy manner. These arcs are always from place ($p \in \mathcal{P}$) to transition ($t \in \mathcal{T}$).

- **Read Arc:** A directed edge (p, t) that inspects the existence of tokens at corresponding input place(s) without consuming them. It is also known as “**Test Arc**” represented with directed arc that ends with a filled circle ($\text{---}\bullet$). Let $I_N(\mathbf{p}_i)$ be the marking at the input place p_i and $I'_N(\mathbf{p}_i)$ be the marking after firing of transition t_j , then formally, it is enabled if:

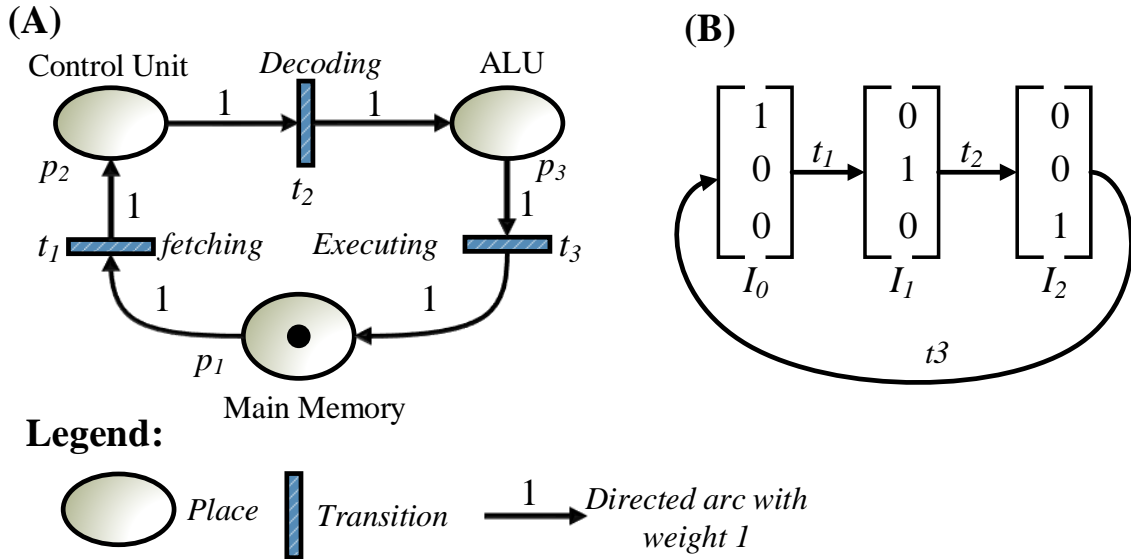


Figure 3.2: An example of Standard Petri Net.

(A) **Simple Petri Net Model of a Machine Cycle:** (i) Fetch Instructions from memory (ii) Control unit decode instructions into commands and send to arithmetic & logic unit (ALU) (iii) ALU then execute commands and store results in main memory. It consist of set of places $\mathcal{P} = \{p_1, p_2, p_3\}$, set of transitions $\mathcal{T} = \{t_1, t_2, t_3\}$, the weight \mathcal{F}_w of each directed arc is 1 and a token at place p_1 means main memory is available. From the initial marking $I_0(1, 0, 0)$, transition t_1 is 1 enabled, t_2 & t_3 are 0 enabled. (B) **The reachability graph of Petri net** obtained from initial state I_0 . This graph of markings consist of vertices and enable transitions on arcs, correspond to tokens at places after firing of transitions. It has one cycle $I_0(1, 0, 0) \rightarrow I_1(0, 1, 0) \rightarrow I_2(0, 0, 1) \rightarrow I_0(1, 0, 0)$ which shows how the system evolves with the repetitive firing sequence $S = t_1 t_2 t_3$. This \mathcal{PN} model is reversible, 1-bounded and deadlock free.

$$I_N(p_i) \geq \mathcal{F}_w(p_i, t_j) \forall i, j \in \mathbb{Z}_{\geq 0} \ \&$$

$$I_N(p_i) \xrightarrow{t_j} I'_N(p_i) \ \& \ I_N(p_i) = I'_N(p_i) \ \text{i.e. the marking remains same}$$


- **Inhibitor Arc:** An inhibitor arc, contrary to read arc, is a directed edge (p, t) that checks absence of markings at corresponding input place(s) without utilizing them. It is generally presented as a directed arc that ends with a small empty circle ($\text{---}\circ$). Based on definition 3, it is enabled if:

$I_N(p_i) < \mathcal{F}_w(p_i, t_j) \forall i, j \in \mathbb{Z}_{\geq 0}$, where $I_N(p_i)$ are markings at input place ‘ p_i ’

and

Number of tokens at place do not change after enabling of transition. i.e.

$$I_N(p_i) \xrightarrow{t_i} I'_N(p_i) \ \& \ I_N(p_i) = I'_N(p_i)$$




- **Equal Arc** is another very useful directed edge that verifies exactly equal tokens at input place, so it ends with two filled small circles (). Formally, a transition is enabled if:

$$I_N(p_i) = \mathcal{F}_w(p_i, t_j) \forall i, j \in \mathbb{Z}_{\geq 0} \ \&$$

$$I_N(p_i) \xrightarrow{t_i} I'_N(p_i) \ \& \ I_N(p_i) = I'_N(p_i)$$

The graphical notations are given in Table 3.1.

Table 3.1: Graphical Notations of directed arcs for Extended \mathcal{PN}

Arc Type	Graphical Representation
Inhibitor Arc	
Read Arc (Test Arc)	
Equal Arc	

3.3 Non Autonomous Petri Nets

With the advancement in technology, the discrete model formalism of Petri net has been evolved. \mathcal{PN} can be synchronized with some external events and/or it can be used to model time dependent systems i.e. system evolves with respect to the conditions associated with the transitions. Hence, these types of Petri nets are said to be non autonomous [54]. Deterministic timed Petri nets and non-deterministic stochastic Petri nets of this class are especially used for performance evaluation of the complex systems that is the prime goal of our work. In section 3.3.1 & 3.3.2, timed Petri nets and stochastic Petri nets are discussed briefly.

3.3.1 Timed Petri Nets

Timed Petri net modeling formalism is used to model and analyze the timing behavior of the system. The time of event occurrence is known a priori so this time is taken as deterministic delay, associated with transitions e.g. customer's waiting time, elapsing time of machine, data transmission time etc.. The transition, once enabled, is fired after this deterministic delay time.

Definition 4 (Timed Petri Net) A timed Petri net is 5-tuple $\mathcal{T}_d\mathcal{PN} = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}_w, I_0, d_t \rangle$, where:

- $\mathcal{P}, \mathcal{T}, \mathcal{F}_w, I_0$ are same as discussed in standard Petri nets definition 3.
- $\mathbf{d}_t : \mathcal{T} \rightarrow \mathbb{Q}_{\geq 0}$ is deterministic time associated with each transition \mathcal{T}_j . This delay time must be elapsed between enabling and firing of a respective transition.

Figure 3.3 illustrates the definition 4 of timed Petri nets.

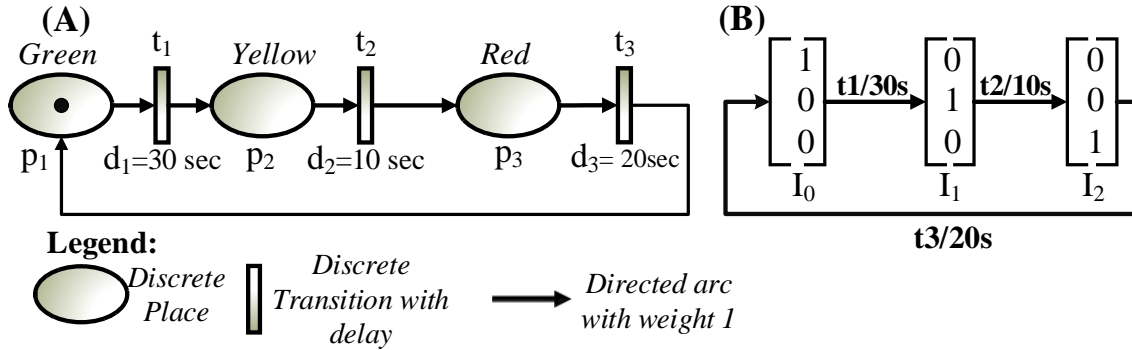


Figure 3.3: (A) Timed Petri net model of Simple Traffic Light Signal.

It consists of a set of places $\{p_1, p_2, p_3\}$ as green, yellow & red light, set of transitions $\{t_1, t_2, t_3\}$ and delay as $t_1 \rightarrow 30s, t_2 \rightarrow 10s$ & $t_3 \rightarrow 20s$. The timing behavior of the system is illustrated through delay transitions. Initially p_1 has one token, meaning the light is green, so transition t_1 is 1-enabled and will be fired after the delay d_1 of 30 seconds. The token is deposited from p_1 to p_2 , indicating the traffic light will turn to amber. Transition t_2 is enabled, but the signal will go to red state after the delay d_2 of 10 seconds has elapsed. Afterwards, t_3 is enabled, but the signal will turn to green (i.e. initial state) after the elapsed time of 20 seconds (d_3). **(B) Reachability Graph.** The graph shows reachable markings after each enabled deterministic transition. The directed arcs show enabled transitions with delay, and resultant markings are shown through 1-D vectors. The graph is cyclic, hence it is live and strongly connected with no deadlock.

Most of the events in real-time systems are random [55], so non-deterministic time delay must be considered to evaluate different behaviors of the system. Hence, a timed Petri net with stochastic timings is presented in section 3.3.2.

3.3.2 Stochastic Petri Nets

Stochastic Petri net (\mathcal{SPN}) is a very significant type of timed Petri net $\mathcal{T}_d\mathcal{PN}$ in which negative exponential distributed firing rates are associated with the transitions [56].

\mathcal{SPN}_s , first introduced in 1982, offers promising benefits even after more than three

decades and has been extensively used to evaluate the performance of complex systems [57, 58, 59, 60, 61]. The non deterministic and independent event occurrence is encountered probabilistically i.e. a random variable D_t that follows exponential distributed time delay with expected delay $\mathcal{E}(D_t)$ of $1/\omega_t$ denoted as:

$$D_t \sim Exp(\omega_t(m), T) \quad (3.1)$$

where $\omega_t(m) \in \mathbb{R}^+$ is the firing rate function that depends upon number of tokens in input place(s) of transitions t_j i.e. $m(p_i) \geq F_w(p_i, {}^o t_j)$ and T represents continuous time such that $\forall T \in [0, \infty)$.

Probability distribution function (**PDF**) and probability density function (**pdf**) of exponential law with rate parameter $\omega_t(m)$ are mathematically given in equation 3.2 & equation 3.3, respectively.

$$P_{D_t}(T; \omega_t(m)) = 1 - e^{-\omega_t(m) \cdot T}, T \geq 0 \quad (3.2)$$

$$p_{D_t}(T; \omega_t(m)) = \omega_t(m) \cdot e^{(-\omega_t(m) \cdot T)}, T \geq 0 \quad (3.3)$$

Furthermore, **memoryless property** is a fundamental characteristic of negative exponential distribution i.e. probability of firing rate (or delay) of random variable D_j at time T is independent of previous behaviors and depends only on current time:

$$Pr[D_j(T_1) \leq T_0 + T_1 \mid D_j(T_1) > T_0] = Pr[D_j(T_1) \leq T_1] \quad (3.4)$$

In \mathcal{SPN} , events occur continuously and independently at random time interval which signifies that it exhibits memoryless property, also known as **Markov property** [56]. Consequently, stochastic systems can be analyzed through continuous-time Markov chain (*CTMC*) [59].

Definition 5 (Stochastic Petri Net) Formally, a stochastic Petri net is defined as 5-tuple $\mathcal{SPN} := \langle \mathcal{P}, \mathcal{T}, \mathcal{F}_w, I_0, \nabla \rangle$, where:

- $\mathcal{P}, \mathcal{T}, \mathcal{F}_w, I_0$ corresponds to places, transitions, weight function and initial marking as illustrated in definition 3 of standard Petri net \mathcal{PN} .
- $\nabla : T \rightarrow \mathcal{R}$ is a stochastic rate function assigned to each transition $t \in \mathcal{T}$, where:
 - $\mathcal{R} \subseteq \mathbb{R}^+$, i.e. stochastic rate belongs to the subspace of positive real numbers ($\mathcal{R} \in [0, \infty)$) and the rate function, as discussed above, obeys negative exponential distribution.

Figure 3.4 demonstrates a stochastic Petri net example to understand definition 5 and to further analyze the semantics of \mathcal{SPN} (3.3.2.1).

3.3.2.1 Semantics of Stochastic Petri Nets.

The stochastic transition rate is associated with its enabling degree \hat{e} defined in equation 3.6. The *firing rate* of transition $t_j \in \mathcal{T}$ that is \hat{e} -enabled with marking 'm' is

defined as the product of its enabling degree at time 't' and rate ω_j associated with transition t_j [54], given as :

$$\omega_j \cdot \mathring{e} = \omega_j(\mathbf{m}) \quad (3.5)$$

$$\mathring{e} = \min_{\mathbf{i} | \mathbf{p}_i \in {}^\circ t_j} \left(\frac{\mathbf{m}(\mathbf{p}_i)}{\mathbf{F}_w(\mathbf{p}_i, {}^\circ t_j)} \right), \text{ and } \mathring{e} \in \mathbb{Z}_{\geq 0} \quad (3.6)$$

where ${}^\circ t_j$ shows the set of input places of transition t_j .

The reachability graph of stochastic Petri net model \mathcal{SPN} is isomorphic to \mathcal{CTMC} , thus, discrete state space in continuous domain is generated through which stochastic timed \mathcal{PN} model will be exhaustively analyzed and verified [62]. \mathcal{CTMC} is homogeneous state-transition graph where each state of chain is linked with the reachable markings in reachability graph of \mathcal{SPN} and probability of transition rates is determined from equation 3.5. Figure 3.4(C) further exemplifies \mathcal{CTMC} of \mathcal{SPN} model.

Generalized stochastic Petri net (\mathcal{GSPN}) is \mathcal{SPN} 's formalism that has immediate transitions along with exponential stochastic transitions extended with priorities, read and/or inhibitor edges [53]. Immediate transitions have always highest priority i.e. once enabled will be fired with zero delay as its firing rate is infinite. Researchers have successfully exploited the power of \mathcal{GSPN} to evaluate the performance of dynamic complex systems [63, 59].

It is worth to mention here that \mathcal{SPN} is the best modeling technique to analyze the non-deterministic behavior of real time computing systems since the automatic state

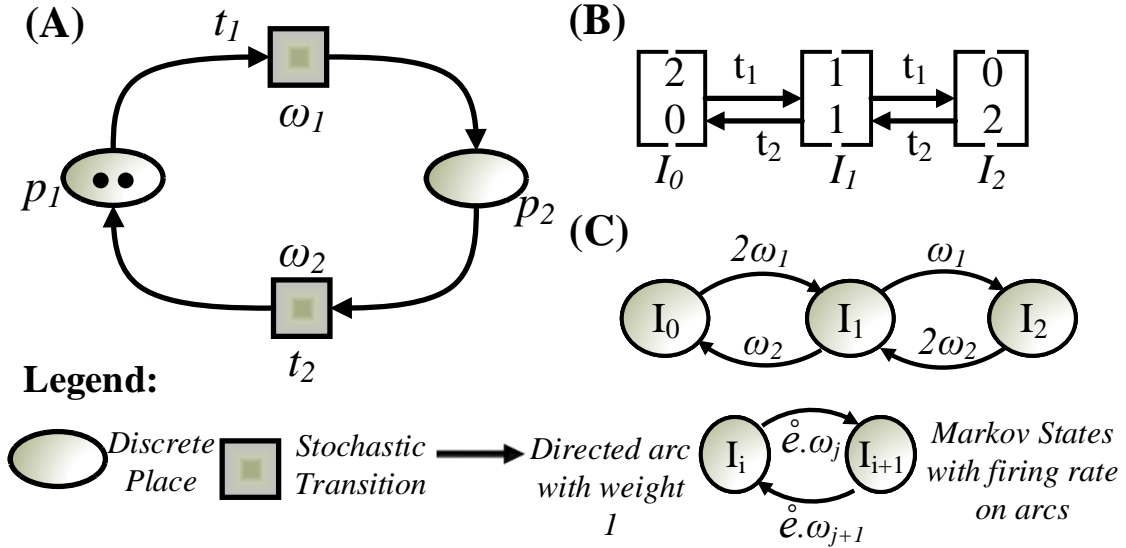


Figure 3.4: Illustration of stochastic Petri net.

(A) SPN Model. Here $\mathcal{P} = \{p_1, p_2\}$, $\mathcal{T} = \{t_1, t_2\}$ with stochastic rates ω_1 & ω_2 , where, \mathcal{P} and \mathcal{T} indicates set of places & transitions respectively with initial marking $I_0 = (2, 0)$ **(B) The Graph of Reachability.** Graph present markings of each place after firing of each enabled transition. **(C) Markov chain.** A Markov process with 3 different states obtained from reachability graph in (B). The directed arcs are labeled with the firing rate obtained from equation 3.5 i.e. product of enabling degree of corresponding transition and its exponential rate e.g. from initial marking m_0 , t_1 is 2-enabled since it is 2-times fire-able and t_2 is 0-enabled so next state m_1 is reached after firing rate of $2 \cdot \omega_1$.

space generation of SPN model simplifies and improves the reliability of complex system's verification process [60]. Despite the fact that Markovian analysis is suitable for bounded SPN model, the graphical model generation of large, complex system is very complicated and thereafter, an infinite state space Markov chain. Consequently, high level Petri net formalism is introduced to overcome the liability of $SPNs$. Colored Petri net (CPN), an abstract high level formalism, is employed to get more structured and compact representation of PNs and further elaborated in Section 3.4.

3.4 Colored Petri Nets Framework

Colored Petri nets (\mathcal{CPN}) was first proposed by Kurt Jensen for model formulation and validation of concurrent systems [64, 65]. The risk of modeling errors due to size and complexity of the model can be reduced momentarily by using these high level Petri nets. In \mathcal{CPN} , standard Petri net is extended by introducing some programming language features like data types known as color sets, color functions, arcs expressions etc. to get more abstract, hierarchical and parameterized model. The copies of identical Place/Transition subsystem are folded to one colored Petri net by applying colors to each subnet with certain conditions without losing the actual behavior of the system. Even in some cases the system with multiple same subnets can be extended just by adding colors. These general principles are applied when converting similar subsystems to color Petri net models. Thus, \mathcal{CPN} modeling significantly simplifies the formal representation as well as analysis of strongly symmetric and large complex systems. Colored Petri nets are formally defined in definition 6.

Definition 6 (Colored Petri Net)

A Colored Petri net is 9-tuple $\mathcal{CPN} = \langle \mathcal{P}, \mathcal{T}, \mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Gamma, \mu, \mathbf{I}_0 \rangle$, where:

- \mathcal{P} & \mathcal{T} are bounded, non empty set of places $\{p_1, p_2, \dots, p_n\}$ and transitions $\{t_1, t_2, \dots, t_m\}$, respectively.
- $\mathcal{A} \subseteq ((T \times P) \cup (P \times T))$ is a set of an ordered pair of edges. Each pair indicates a directed arc from place to transition (p, t) or from transition to place (t, p) and

$$(p, t) \neq (t, p)$$

- Λ is a non-empty set of color sets analogous to data types. Each place belongs to only one type of color-set. The data types for color-sets have two classes [66]:

(i) **Simple data types:** It includes dot, integer, boolean, string, enumeration and index.

(ii) **Compound data types:** It is a tuple based on simple data types. e.g. Product and Union.

- $\Omega : \mathcal{P} \rightarrow \Lambda$ is a color function that maps each instance of place $p \in P$ to colorset $\Omega(p) \in \Lambda$.
- \mathcal{E} is set of expressions made-up of constants, variables, functions, operation symbols of colorsets etc.. An expression is used to represent arcs inscription, guard function and/or to define initial marking.
- $\Gamma : \mathcal{A} \rightarrow \mathcal{E}$ is an arc expression, assigned to each arc $a \in \mathcal{A}$ and evaluates to a single element or a multiset (definition 7). It determines which type of tokens can flow over the arc and thus color set of the arc expression must match with the color set of the place attached to the arc.
- $\mu : \mathcal{T} \rightarrow \mathcal{E}$ is a guard function i.e. a constraint on each transition such that $t \in \mathcal{T}$ and \mathcal{E} is of boolean type evaluates to *True* or *False*. Firing of transition is allowed only if the guard is satisfied.

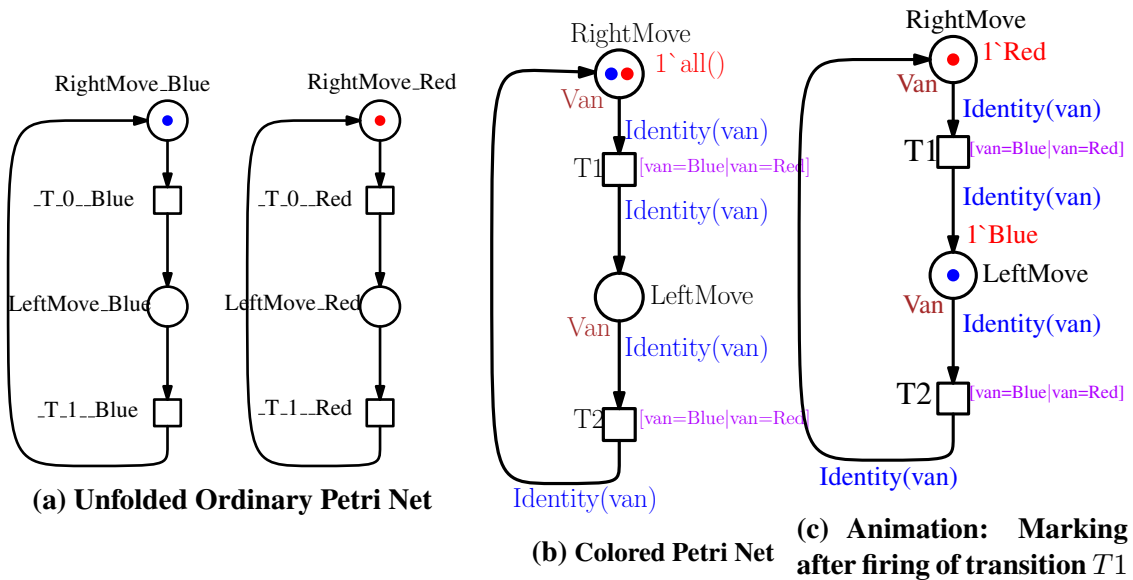
- $I_0 : \mathcal{P} \rightarrow \mathcal{E}$, is initialization function assigned to each place $p \in \mathcal{P}$. \mathcal{E} evaluates multiset of tokens with colors that corresponds to colorset of the place ($\Omega(p)$). Multiset addition operator ($++$) is combined with multiset operator (\wedge) to briefly represent the markings of a place. For example, a boolean multiset "**3`true++2`false**" presents 3 instances of true and 2 instances of false, with total of 5 boolean values.
- $\mathcal{P} \cap \mathcal{T} \cap \mathcal{A} = \varphi$ & $\mathcal{P} \cup \mathcal{T} \cup \mathcal{A} \neq \varphi$ i.e. places, transitions and arcs are independent sets. Figure 3.5 demonstrate semantics of \mathcal{CPN} (definition 6). The example presents Qualitative Colored Petri net model of a system consist of two vans Blue and Red that have similar behavior i.e. moving from left to right and then back from right to left continuously. They have same structure (Figure 3.5a) thus folded to its equivalent colored Petri net(3.5b, 3.5c) by defining colors with certain constraints ([54], Chapter 01).

Definition 7 (Multiset): A multi-set is a generalized form of set which represent several occurrences of the same element. The number of times an event occurs is its multiplicity or co-efficient and cardinality is the total elements in the set.

Formally, it is defined as a pair (\mathcal{X}, γ) where \mathcal{X} is a nonempty set of finite elements and $\gamma : \mathcal{X} \rightarrow \mathbb{Z}_{\geq 1}$ is a mapping which associate each element of \mathcal{X} to the set of natural numbers. For instance in Figure 3.5b, $\mathcal{X} = \{blue, red\}$ is a color set and $\gamma = 1`red ++ 1`blue$ is a multi-set over \mathcal{X} means 1 occurrence of blue van and 1 event of red van, i. e. $\gamma(blue) = 1$ & $\gamma(red) = 1$ whereas cardinality of \mathcal{X} is 2.

Table 3.2: Declarations for the \mathcal{CPN} model of Figure 3.5

Declarations
Colorset Van = enum with Blue, Red;
Variable van : Van;
Function Van Identity (Van van1) [van1=Blue]1`Blue++[van1=Red]1`Red;

**Figure 3.5: Folding and Unfolding of Colored Petri Nets Example**

(a) \mathcal{PN} model of a system comprises of two Vans (blue & red) exhibiting same structure, modeled with two isolated subnets. (b) Folding of Simple Petri net model to Colored Petri net by defining colors with declarations given in Table 3.2. Two vans are represented as one colored place and variable van is defined with color-set/data-type Van also known as binding. Identity function with variable ' van ' is defined as arcs expression to distinguish the movements of vans accordingly. Each transition is associated with boolean expression i. e guard over defined variable ' van ' and transition is enabled only if guard evaluates to *true*. Initial marking is $1`all()$ means there is 1 occurrence of each color of Van . (c) From the initial marking $T1$ is 1-enabled for blue van, 1-enabled for red van and $T2$ is 0-enabled. Both vans have same probability of moving hence, randomly chosen, state of net indicates one blue van ($1`Blue$) will move from right to left after firing of transition $T1$ once.

3.4.1 Colored Stochastic Petri Nets

Colored stochastic Petri nets (\mathcal{SPN}^c) are the colored extension of stochastic Petri nets (Section 3.3.2) and have tremendous modeling power to present large scale complex systems. "Colors" are used to identify tokens where arcs expressions & guards have same semantics as that of standard \mathcal{CPN} (Definition 6) [67, 68].

Definition 8 *A Colored stochastic Petri net is a tuple $\mathcal{SPN}^c = \langle \mathcal{P}, \mathcal{T}, \mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Gamma, \mu, \mathbf{I}_0, \nabla, \lambda \rangle$, where:*

- $(\mathcal{P}, \mathcal{A}, \Lambda, \Omega, \mathcal{E}, \Gamma, \mu, \mathbf{I}_0)$ are same as defined in standard \mathcal{CPN} (6)
- $\mathcal{T} \in \mathcal{T}_{stochastic} \cup \mathcal{T}_{immediate} \cup \mathcal{T}_{timed}$ i.e. it consists of transitions with exponentially distributed firing rates ($\mathcal{T}_{stochastic}$), transitions with zero delay ($\mathcal{T}_{immediate}$) and transitions with deterministic delay (\mathcal{T}_{timed}).
- $\nabla : \mathcal{I}(\mathcal{T}_{stochastic}) \rightarrow \mathcal{R}$ is stochastic rate function assigned to each colored instance of stochastic transition 't(c)' where $\mathbf{t}(\mathbf{c})$ corresponds to the binding which after unfolding will become an uncolored \mathcal{SPN} transition, $\forall \mathbf{t}(\mathbf{c}) \in \mathcal{I}(\mathcal{T}_{stochastic})$ & $\mathcal{R} \subseteq \mathbb{R}^+$.
- $\lambda : \mathcal{I}(\mathcal{T}_{timed}) \rightarrow \mathbb{R}^+$ is deterministic non negative delay assigned to each colored instance of timed transition $\forall \mathbf{t}(\mathbf{c}) \in \mathcal{I}(\mathcal{T}_{timed})$.

3.5 Formal Verification

Formal verification is like “**bug hunting**”[69] i.e. It searches for errors in the system and gives counter example if error comes. Formal analysis techniques, as opposed to simulation-based testing, provide deeper insights into the system model [70], and are also essential for safety-critical systems in which most of the events are probabilistic. Hence, characterization of these events through formal analysis techniques gives proof of reliability and guarantees the performance of real time system. Model checking is one of the most extensive automatic and efficient formal verification technique for validation of concurrent systems [71, 72].

3.5.1 Model Checking

The pioneers of model checking techniques, Clarke and Emerson, introduced a very efficient formal method technique for verification of the system in 1980’s [73]. Model checking is a rigorous method that exhaustively explores the whole reachable states of the system to check the correctness of system properties written in some formal specification language e.g. linear-time temporal logic (**LTL**), computational tree (or branching-time) logic (**CTL**) or probabilistic computational tree logic (**PCTL**) etc. [74]. If there is a violation of the property, model checker gives counter example i.e. identifies the path in which property is not satisfied and in this way, the model can be updated continuously. Thus this guarantees the correctness of the system. Figure 3.6 demonstrates the process of model checking.

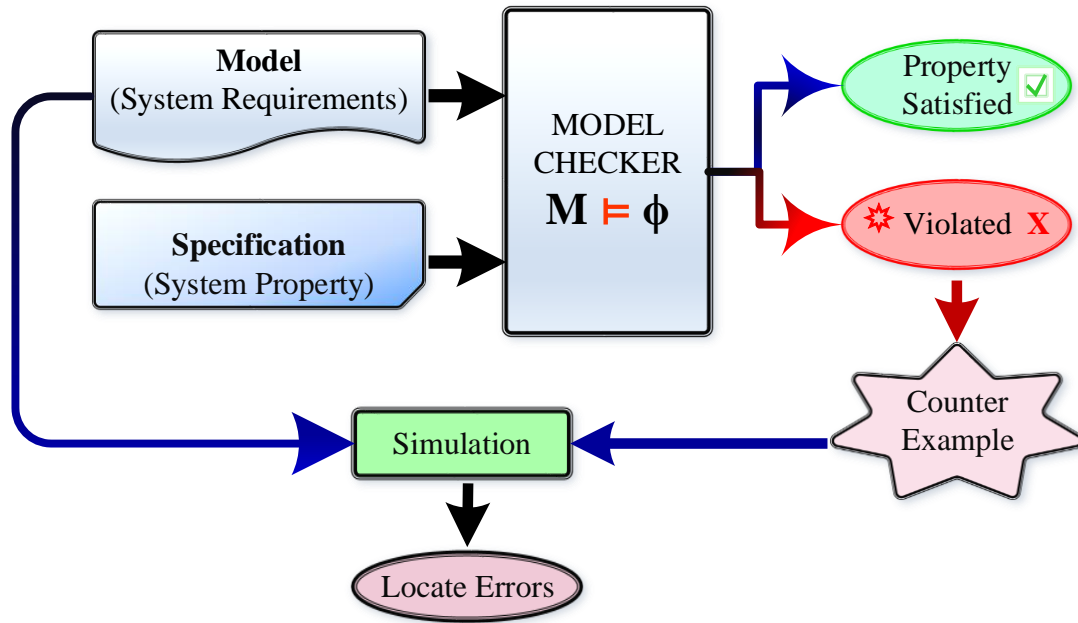


Figure 3.6: Model Checking. In model checking technique, the model checker takes the formal model of the system along with its formal specifications and produce results in either of two forms: *(i)* True i.e. property is fulfilled. *(ii)* False i.e. property is violated and in this case it also generates counter example based on which we identify the errors, refine the model and repeat the process. Hence, it gives the proof that the system is in good condition.

3.5.2 Probabilistic Model Checking

The randomness in real world problems demands a very efficient and reliable method that will guarantee the performance of the system. Probabilistic model checking is introduced for automated quantitative analysis and verification of inherently probabilistic systems. It models the informal specifications of system to formal specifications. The stochastic model checker takes the stochastic transition state model and probabilistic temporal properties as an input. Its output results are twofold: (a) it checks whether the property is True or False (Verified/ Not-verified) and (b) it can also provide quantitative

results in terms of probability or the expected reward time [75]. Moreover, if the property does not satisfy then it indicates a path that leads to an exceptional anomaly state (also known as Counter Example). The whole probabilistic model checking process is illustrated in Figure (3.7).

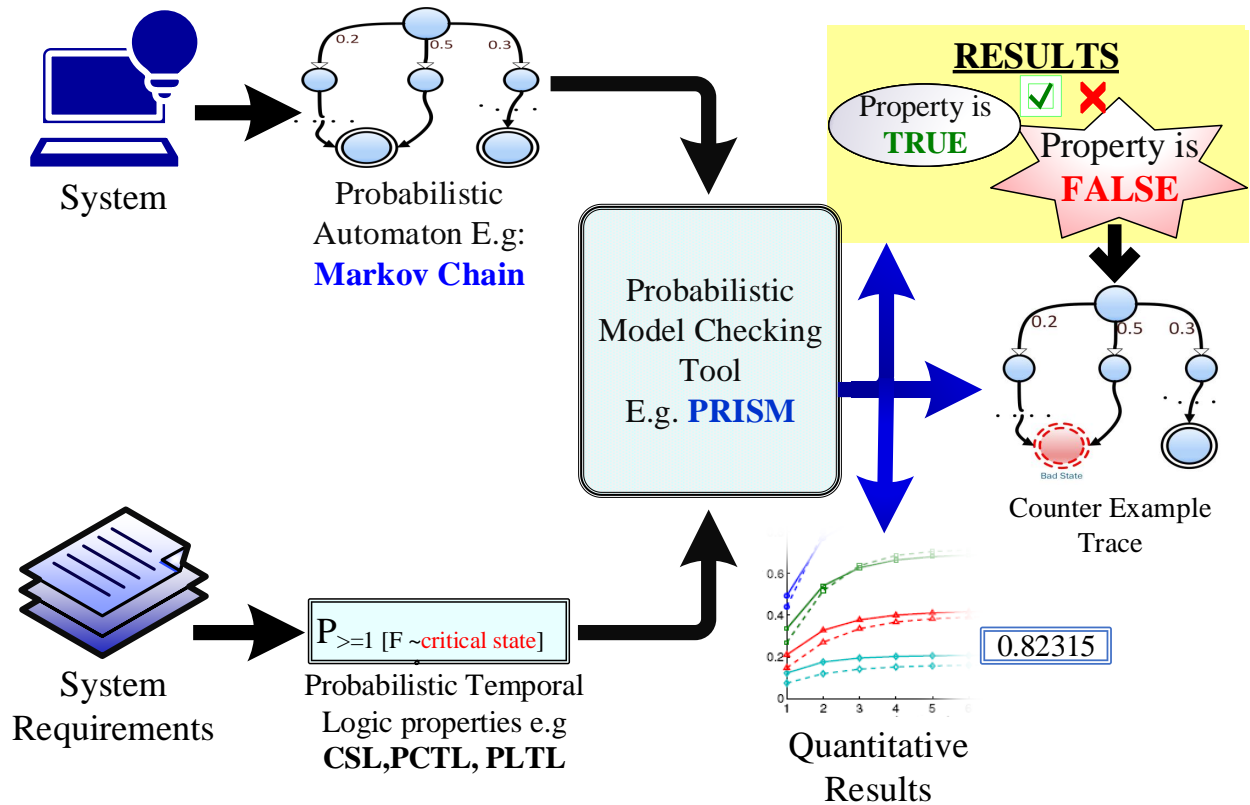


Figure 3.7: Probabilistic Model Checking Process.

A stochastic state transition model and probabilistic temporal specifications, according to systems requirements, are given to stochastic model checker. Its output results are of two types: (i) either the property is true or false (If the property is false, it indicates the path of bad state.) (ii) it can also generate numerical results e.g. expected time of state or likelihood of event occurrence etc..

Continuous-time Markov chains (CTMCs) are used to model the stochastic behavior of real-time systems and are commonly used for the performance analysis of nonde-

terministic real-time systems. In contrast to discrete time-step for DTMC (discrete time Markov chain), the event/transition occurrence in CTMC is in real time [35]. In CTMC model, states are discrete and transitions between states occur with exponentially distributed rates. It is formally defined in definition 9. A state-of-the-art probabilistic model checking tool, PRISM, is used in this work for schedulability analysis of stochastic real time systems. PRISM takes CTMC model of the system along with the temporal formal specifications written in continuous stochastic logic (CSL). CTMCs are best-known to adequately portray the stochastic real-time systems as well as economical solutions for computation of performance analysis and verification of the system [76].

Definition 9 (Continuous-time Markov chain) *A labelled Continuous-time Markov chain is a quadruple [35], $CTMC = (S_t, s_o, T_R, \mathcal{L})$*

- S_t is a finite state space
- $s_o \in S_t$ is the initial state in the state space
- $T_R : S_t \times S_t \rightarrow \mathbb{R} \geq 0$ is the stochastic negative exponential rate matrix function over state space S_t . It indicates that the probability to stay in state s maximally t time units is $1 - e^{-T_R(s) \cdot t}$. It means if $T_R(s, s') = \omega$, then the average speed from s to s' is $1/\omega$.
- $\mathcal{L} : S_t \rightarrow 2^{AP}$ is a label function that assigns a set of atomic propositions logic ('AP') to each state of state space S_t .

The implicit stochastic process in \mathcal{SPN} is a $CTMC$, discussed above in section 3.3.2.1. Therefore, CTMC models of \mathcal{SPN} and \mathcal{SPN}^c are defined in PRISM to analyze their probabilistic behavior. The PRISM modeling is based on reactive modules [77], finite state variables and guards/transition rules. The guarded commands express the evolution of states, labeled with rates (or probabilities in DTMC) and actions. Its syntax is specified as:

$$[action] \text{ guard } \omega_1 : \mu_1 + \dots + \omega_n : \mu_n \quad (3.7)$$

where label *actions* are for synchronization of transitions, *guard* represents predicate over variables, ω_i expression is the stochastic rate assigned to transitions and μ_i expression shows the updated values of state variables only if the guard is satisfied. Quantitative properties are specified by extending CTMC through reward structures, expressed as:

$$[action] \text{ guard } : \gamma_w \quad (3.8)$$

Here *action* labels are optional and γ_w is the expression of reward assigned to the state variables when guard evaluates to true.

3.5.3 Continuous Stochastic Logic Model Checking over CTMC

The probabilistic temporal logics are the developed techniques for performance evaluation and verification of systems [76]. CSL is the probabilistic representation of CTL (computational tree logic) for properties encoding of CTMC. It uses probabilistic logic

quantifiers rather than standard path quantifiers. The properties of underlying stochastic state-transition model are formally specified in CSL which also provides numerical methods represented as rewards [78]. CSL syntax is formally defined by:

$$\Psi ::= TRUE \mid AP \mid \sim \Psi \mid \Psi \vee \Psi \mid \mathbb{S}_{\leq p}(\Psi) \mid \mathbb{P}_{\leq p}(\varphi) \mid \mathbb{R}_{\leq \gamma}(F \Psi)$$

$$\varphi ::= X \Psi \mid \Psi U^I \Psi$$

Here AP is an atomic proposition, comparison operator $\leq \in \{<, \leq, \geq, >\}$, probability $p \in [0, 1]$, reward $\gamma \in \mathbb{R}_{\geq 0}$ and I is the time interval of $\mathbb{R}_{\geq 0}$.

- $\mathbb{S}_{\leq p}(\Psi)$ formula indicates the steady-state probability of Ψ being true is $\leq p$.
- $\mathbb{P}_{\leq p}(\varphi)$ represents probability of path formula φ being satisfied is $\leq p$. The temporal path formula operators are: X (*neXt*), F (*in future/eventually*), U (*Until*) or $U^{\leq t}$ (*Bounded Until*), G (*always satisfied/globally true*), W (*Weak Until*) and/or their complex combinations.
- $\mathbb{R}_{\leq \gamma}(F \Psi)$ expression specifies the expected reward rate. Reward-based properties provide quantitative performance evaluation based on states and/or transitions of CTMC model [79].

The numerical analysis of properties is done by altering comparative bound in \mathbb{S} , \mathbb{P} and \mathbb{R} expressions with $=?$.

The CSL model verification problem is decidable [80]. Let \mathcal{M} be a CTMC model, s is the state in \mathcal{M} and Ψ represents *CSL* temporal formula against CTMC states, then the stochastic model checking problem is to find out whether the property Ψ satisfies in state s of the model \mathcal{M} . Mathematically, it is represented as $s \models_{\mathcal{M}} \Psi$.

The non-deterministic behavior of a real-time scheduling problem is modeled through CTMC based on *SPN*. CSL properties are specified over different states of CTMC in order to verify the correctness of simulation results and performance evaluation. A CTMC model of general job scheduler (Figure 4.1) is given in the Appendix A. The model starts with the keyword *ctmc* that indicates the type of probabilistic model analyzed in this study. It comprises of state variables and constants. The guarded commands represent behavior of module and follow the syntax as defined in equation 3.7. A symbol \rightarrow is used to isolate the guarded transition rules and their exponential rates. The state variables are updated to new values (s') when corresponding transition rules satisfy. The module ends with the keyword *endmodule*. The reward rules are defined in reward structure, enclosed in the keywords *rewards* and *endrewards*. Various behavioral properties are encoded in CSL over CTMC model. The CSL temporal stochastic properties against CTMC model are formally defined and elaborated in section 5.3.

3.6 Software Environment for Modeling & Verification

In this research work, an extensive unifying Snoopy tool [81] is used for Petri net modeling and performance evaluation of real time scheduling via multiple simulations.

The behavioral and structural properties of Petri nets are analyzed through Charlie tool [82, 83]. Moreover, continuous time Markov chain (CTMC) of Petri net model is analyzed in symbolic model checker PRISM [84] to further verify the scheduling properties and quantitative analysis of the system [75].

Chapter 4

Model Formulation

This chapter formulates two important formal models used in this study. Stochastic Petri net (SPN) model of general job scheduling is elaborated in section 4.1. In section 4.2, SPN^c model of global earliest deadline first ($gEDF$) scheduling algorithm is demonstrated in detail.

4.1 SPN Model of General Job Scheduler

Real time job scheduler handles the real time workload in such a manner that all the submitted tasks are executed following their timing constraints and it extremely depends on available resources. Moreover, a job scheduler must also be satisfied enough to manage the ergodic occurrence of real time events. Thus, a feasible Job scheduler is necessary for dynamic workload management to satisfy the ever increasing computing

demands [85]. Thereupon, a very efficient non-deterministic SPN modeling formalism as discussed in section 3.3.2, is applied to determine the dynamic behavior of a general job scheduler (Figure 4.1). Petri net places are used to express the states of the system e.g. Task is dormant (in **JOB_POOL**), in **Ready** state, **Run** state, **Waiting** or completed (**Tasks_Completed**) and transitions represent events/conditions at random time interval upon which system goes from one state to another whereas markings of a place shows the evolution of system.

Suppose initially ‘ N ’ independent tasks are submitted by the user to **JOB_POOL** and ‘ M ’ number of **Resources/processors** are available. The inter-arrival time of tasks follows exponential distribution function represented here with stochastic transition **Task_Arrives**. The description of each place and transition is given in Table 4.1 and Table 4.2. The scheduler always schedules the active tasks, if the processor is available, based on some scheduling policy. The process will repeat itself once N jobs are completed such that $JOB_POOL = \phi \wedge TasksCompleted \geq N$, depicted in Figure 4.1 as inhibitor arc and read arc. The simulation plots of this model are presented in Figure 5.1 and Figure 5.2. Furthermore, SPN ’s reachability graph that is isomorphic to CTMC and verification of various probabilistic properties of model are also done in Section 5.3. Different experiments can be performed on this model according to the system requirements e.g. by changing the values of N and M . Thus this model provides a basis to automate real time workload management and analysis based on different parameters.

If these N real time jobs are to be processed, in parallel, by ‘ M ’ number of processors using real time scheduling algorithms then SPN model becomes very large and complex. So in order to overcome this limitation, colored Petri net, a compact modeling technique, is exploited and demonstrated in the next section.

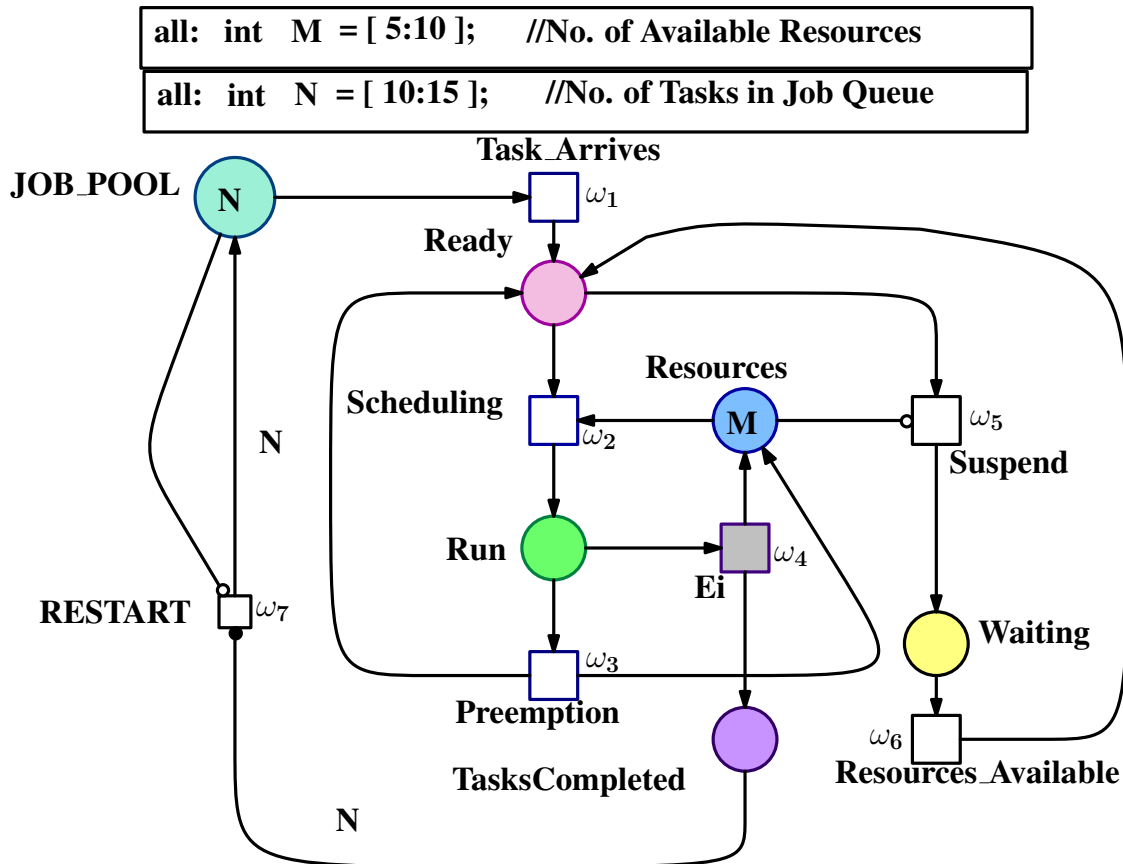


Figure 4.1: SPN Model of Job Scheduler.

An SPN model comprises of set of places $\mathcal{P} = \{JOB_POOL, Ready, Run, Resources, Waiting, TasksCompleted\}$, set of transitions $\mathcal{T} = \{Task_Arrives, Scheduling, Preemption, Ei, Suspend, Resources_Available, RESTART\}$ with initial marking $m_0 = (N, 0, 0, M, 0, 0)$ where N and M are declared as constant values for initial number of tasks and idle processors respectively. Each directed arc has a weight ‘1’ except input arc from RESTART to JOB_POOL that has a multiplicity of ‘ N ’ which indicates ‘ N ’ new Tasks will be submitted to the JOB_POOL when transition RESTART becomes enabled. Here the multiplicity of read arc is also ‘1’.

Table 4.1: Description of Places in Figure 4.1

Place	Description
JOB_POOL	Total number of tasks in Job Queue
Ready	Task is ready for execution and waiting to be assigned to the processor
Resources	Tokens at this place show the number of available resources or idle_processors with equal computing power
Run	Task in running state means that it has been assigned to processor for execution.
Waiting	Active task is suspended as all the processors are busy and it has to wait for processor to get free.
TasksCompleted	Task is completed after elapsing random execution time (E_i)

Table 4.2: Description of Transitions in Figure 4.1

Transition	Description	Rate function
Task_Arrives	Real time jobs are released with firing rate of ω_1 and jobs are now ready for execution	ω_1
Scheduling	It indicates that tasks are scheduled based on some scheduling policy constraints and processors are assigned to them with rate ω_2	ω_2
Preemption	A running task may be preempted by some high priority task. This event occurs for dynamic scheduling algorithm.	ω_3
Suspend	This transition is enabled when all the processors are busy and the task has to wait for a free processor.	ω_5
Resources_Available	Waiting task goes to ready state on the availability of resources and is re-prioritized.	ω_6
E_i	The random occurrence of each computational task has exponentially distributed execution time (E_i) with rate ω_4 after which the task is completed and processor is released at the same time.	ω_4
RESTART	The process repeats itself after completion of N tasks submitted by user.	ω_7

4.2 Colored Stochastic Petri Net of $g\mathcal{EDF}$ Scheduling

Algorithm

A powerful and expressive modeling formalism of colored stochastic Petri nets (SPN^c) is proposed to deeply understand the probabilistic nature of real time scheduling algorithms. SPN^c modeling technique is robust enough to manage the strategy changes in real time applications. In this study, $g\mathcal{EDF}$ multiprocessor scheduling algorithm is evaluated using SPN^c . A work flow of $g\mathcal{EDF}$ is demonstrated above in Figure 2.3 and its SPN^c model is illustrated in Figure 4.2.

The following assumptions are taken into account while SPN^c model formulation of $g\mathcal{EDF}$:

- (A1) All the processors are homogeneous i.e. each processor has equal computing power and once the task execution is preempted, it can resume on any other idle processor.
- (A2) Multiple jobs are manipulated in this model. All the jobs are independent and each job has only one task.
- (A3) All the tasks are Sporadic (i.e. Asynchronous tasks with minimum inter-arrival time and hard deadlines [86]).
- (A4) All the tasks have static deadline that is defined a priori and scheduling policy is applied immediately once the job releases.

- (A5) For simplicity of the model, deadline time is represented in integers and no two tasks have same deadline.
- (A6) For experimental purposes, number of Tasks and Processors are deliberately limited.
- (A7) The preemption cost, context switches and systems overheads are assumed to be negligibly small, thus ignored.

All the declarations of SPN^c model of $gEDF$ are given in Table 4.3. SPN^c places are associated with color-sets that represent different states. Each place must have tokens of one color type and all the color instances have either same markings depicted here as **all()** or different markings that can be assigned by applying predicates. Furthermore, stochastic transitions have random *rate functions* as well as *guards*. The stochastic rate may also be associated separately with each color instance and the transition is fire-able when the guard expression evaluates to “*true*”. The system evolves on the basis of markings that depends on corresponding arc’s expression (formulated from variables and constants) and enabling transition(s). Multiple tasks are distinguished on the basis of colors as in the place **Job_Pool** that is defined as a 2-tuple **Product** color-set (**TaskId, Deadline**) to define each task. The set of places and transitions are demonstrated in Table 4.5 and Table 4.6. The directed arcs in SPN^c have expressions instead of integer numbers in standard Petri nets. The expression evaluates to a multi-set that must have color domain of its associated place. For example, in Figure 4.2 the

arc expression from coarse transition **Check** to place **Task_IN_Ready** is:

$$[\text{new_DL} \geq \text{DL}]1'(\text{NewId}, \text{new_DL})$$

It implies if the condition $[\text{new_DL} \geq \text{DL}]$ fulfills, '1' token of product type (NewId , new_DL) will be added to place Task_IN_Ready which also has product color type. Here new_DL , NewId & DL are variables of type Deadline, TaskId & Deadline as shown in Table 4.3. The rest of the arcs expressions are determined in the same manner. Moreover, the active task with the shortest absolute deadline is always scheduled first by the scheduler specified through transitions '**Check_Priorities**', '**New_Priorities**' and macro transition '**Check**'(see Table 4.6).

Table 4.3: Declarations for the \mathcal{SPN}^c model of Figure 4.2

Declarations

Colorset ID = int with 1-N; Colorset Deadline = int with 0-10; Colorset Processors = int with 1-10;
 Colorset Count = int with 1-10;
 Colorset Status = bool;
 Colorset Dot = dot;
 Colorset T_IDXDeadline = product with ID, Deadline;

Variable TaskId : ID;
 Variable DL : Deadline;
 Variable new_DL : Deadline;
 Variable z : T_IDXDeadline;
 Variable NewId : ID;
 Variable s : Count;
 Variable dl : Deadline;

Constant N = int with 10;
 Constant Total_Tasks = int with 10;
 Constant Timer = int with 0;

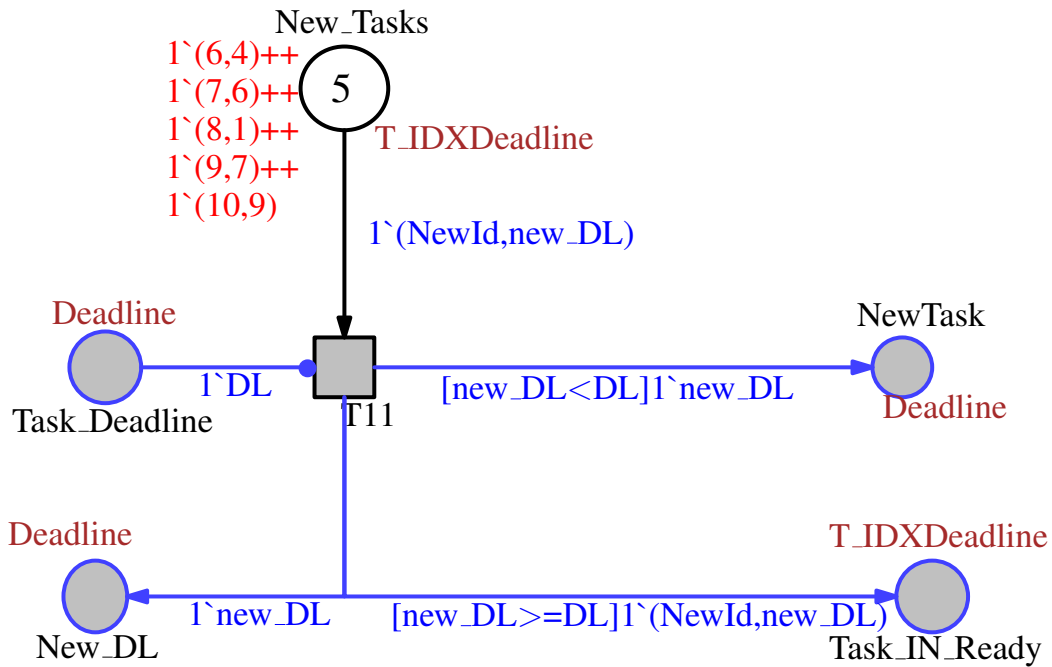


Figure 4.3: Check Module.

Hierarchical structure behind Macro Transition “Check”. Place **New_Tasks** is declared as *Colorset* $T_IDXDeadline = product\ with\ ID,\ Deadline$ (Table 4.3) with initial markings explained in Table 4.4. Blue part of sub-net shows its association with neighboring elements from super-net (immediate higher level) viz. Figure 4.2

The extensive attributes of Petri nets (Section 3.2.1) are applied to the SPN^C model of $gEDF$ as well. For example, “*Read Arc*” from Place **Task_Deadline** to **Check** labeled with $1\ DL$ that reads the task deadline to compare with the deadline of new task and the place marking remains same. “*Inhibitor Arc*” from **Procs_Idle** to **Suspend** transition is there in order to check when all the processors become busy. “*Equal arc*” from the place **Completed_Tasks** with arc inscription $Total_Tasks\ \dot$ enables the transition, if tokens at place **Completed_Task** exactly equals to arc’s multiplicity $Total_Tasks$ (Total number of tasks submitted) which implies all the tasks are executed successfully.

The graphical representation of \mathcal{SPN}^c model is simplified and much more understandable by utilizing following advanced features:

- **Macro Nodes:** Macro Nodes (also known as coarse nodes) are used to define the hierarchical representation of the Petri net model [81]. Even larger \mathcal{PN} models can be designed more systematically using coarse transitions and coarse places. **Macro-Transition** is enclosed with a unique transition surrounded by a sub-net. It is linked with super-net (one level high in hierarchy) whereas **Macro-Place** is a place bounded subnet and associated with the super-net. In Figure 4.2, only macro-transitions are used e.g. “*Check*” and “*CLOCKS*” are coarse-transitions.
- **Logical Nodes :** The graphical duplicates of places and transitions are known as logical nodes, used to improve the model readability. They have logically same interpretation and just different physical identities. These nodes are also commonly known as **fusion nodes**. In this study, logical places and transitions are represented with filled Grey color (Figure 4.2, Figure 4.3).

4.2.1 CLOCKS MODULE

The hierarchical structure behind coarse transition “**CLOCKS**” in Figure 4.2 continuously checks the deadline of each task (Figure 4.4). It is like a deadline timer that eventually begins when the Task with highest priority starts execution after the processor is assigned to it. In Figure 4.5, only one clock of Task (1, 2) (1st element is ID followed by its deadline) is illustrated. All other clocks illustrated in Figure 4.4 are

working on the same principle.

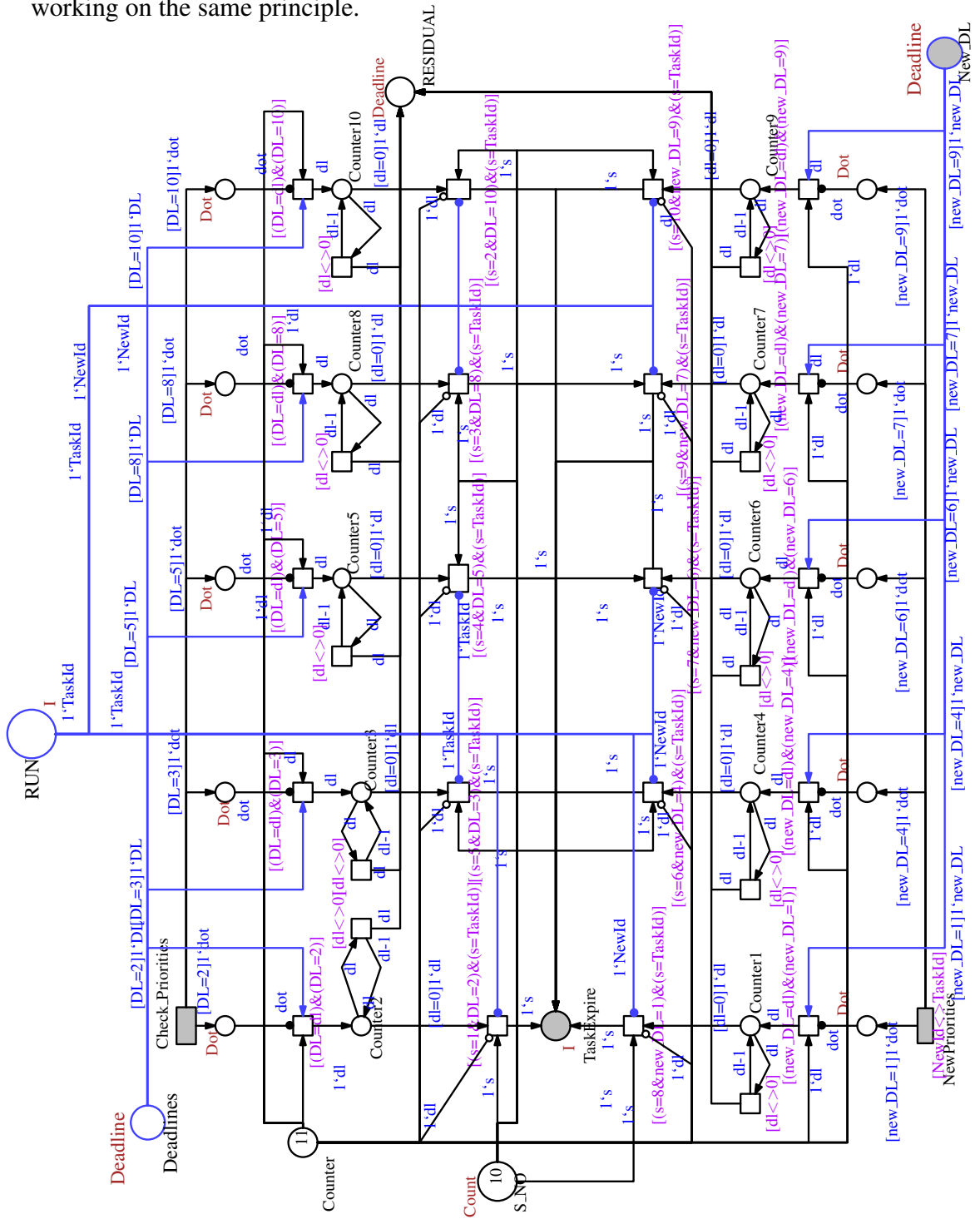


Figure 4.4: CLOCKS Module.
Hierarchical Model behind Coarse Transition “CLOCKS”

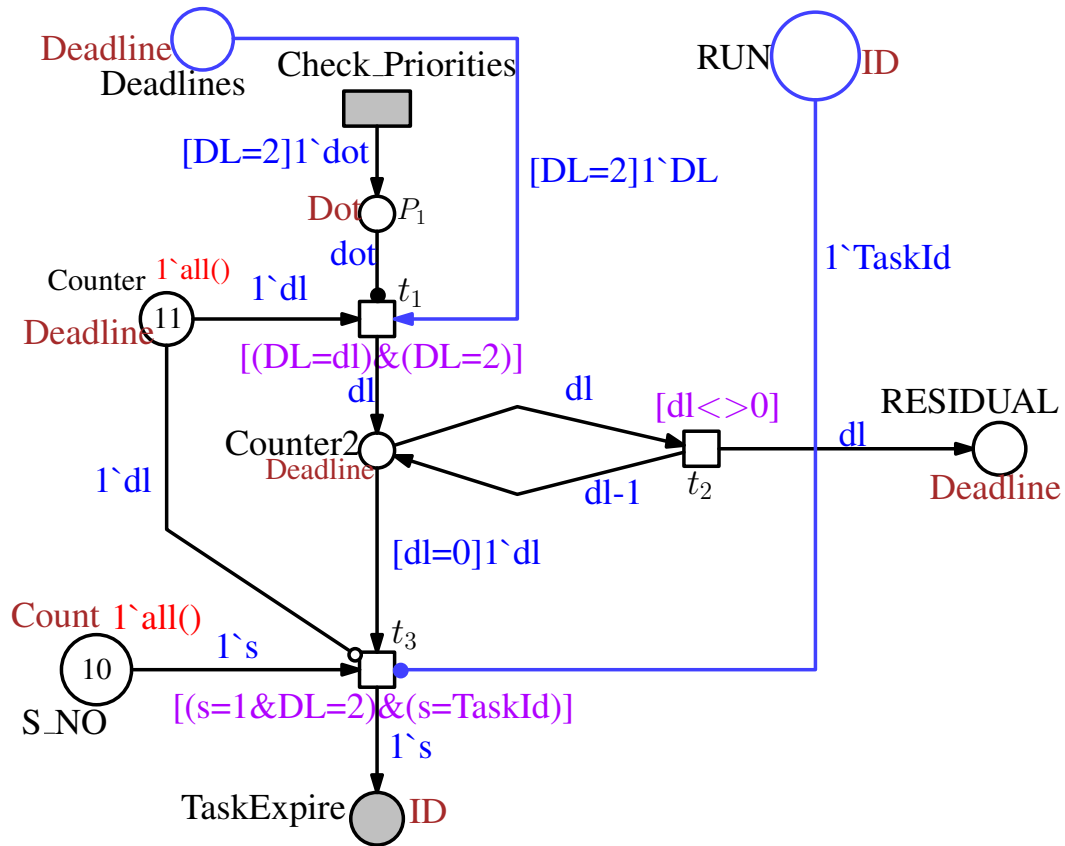


Figure 4.5: Hierarchical Model behind Coarse Transition “CLOCKS”.

The clock for Task_Id = 1 and Deadline = 2 $\{(1,2)\}$ from Figure 4.4 is shown here. The blue part of the net shows its link to super-net (Figure 4.2) and the logical nodes are colored in Grey. The declarations are: *Colorset ID = int with 1-N*; *Colorset Deadline = int with 0-10*, *Colorset Count = int with 1-10* and *Variable TaskId : ID*, *Variable DL : Deadline*, *Variable s : Count*, *Variable dl : Deadline* and *Constant N = int with 10*

Figure 4.5 shows that when the task with deadline $DL = 2$ starts execution, transition $t1$ is enabled and produce output token at *Counter2* place. This eventually will start the decrement counter by enabling transition $t2$ i.e. *while (dl \neq 0) {dl- }* and also produces token at output place “*RESIDUAL*” shown with the arc expression *dl*. But if deadline reaches 0 and task is still in running state then transition $t3$ becomes

fire-able which indicates that the processing of task is not completed within deadline and it will be expired. It is specified in the model (Figure 4.5) with the transition $t3$ which reads $Task_Id$ from place “ RUN ” (shown with read arc) and compares it with the serial number s of deadline that reaches to zero. This is formulated with arc expression $[dl=0]I`dl$ and guard condition $[(s = 1 \& DL = 2) \& (s = TaskId)]$. The “true” result of guard condition will add a token $1`s$ to $TaskExpire$ that shows task does not complete within deadline and thus system fails. Here the variable dl is used to store residual time to deadline and variable s is used as a serial number to identify $Task_Id$. It also evaluates remaining time to deadline (dl) of each task by depositing token into the $RESIDUAL$ place that is linked to the transition $PREEMPTION$ in super-net to determine when the task should be interrupted. The clock is evolving with rate 1 (i.e. $\frac{d_x}{d_t} = 1$).

This is how clocks, the fundamental element of real time scheduling systems, are functioning at the back-end for each task.

In summary, SPN^c model in Figure 4.2 is able to cope with the dynamic behaviors of hard real time tasks. A number of experiments can be performed very efficiently on this model to explore the desired performance parameters via multiple simulations presented in section 5. More importantly, in order to schedule arbitrary ‘N’ different tasks using $gEDF$ scheduling algorithm, there is no need to modify structure of the model instead just additional colors are to be added and guards/predicates are modified i.e. adding color means another subnet. Additionally, SPN^c model can also be unfolded

to uncolored stochastic Petri net model e.g. the flat SPN model of Figure 4.2 comprises of 781 places, 13,936 transitions, 47,247 standard arcs, 13,265 read arcs, 735 inhibitor arcs and 1 equal arc. It thus provides sufficient analysis schemes for model validation and verification of valuable distinct system attributes.

Table 4.4: Initial Markings defined in SPN^c model illustrated in Figure 4.2

Place	Marking	Description
Job_Pool	$1(1,2)++$ $1(2,10)++$ $1(3,8)++$ $1(4,5)++$ $1(5,3)$	Different jobs are in Job_Pool ab-initio. Each job has a task with unique ID and deadline e.g. $1(2,10)$ means one occurrence of taskID '2' with deadline '10 time units'. Here '++' is the multi-set operator that combines multiple markings. These are the example markings that can be modified according to the system requirements.
Procs_Idle	$4\text{all}()$ or 4dot	It defines the presence of available resources initially. Marking function all() represents all the colors in a color-set have same number of tokens. Here " $4\text{all}()$ " means 4 tokens of color-set <i>Dot</i> or presence of 4 initial processors. Color-set 'Dot' is used to indicate all the processors are homogeneous. Initial markings can be altered according to the experiments performed.
New_Tasks	$1(6,4)++$ $1(7,6)++$ $1(8,1)++$ $1(9,7)++$ $1(10,9)$	During execution of tasks, new jobs may arrive depicted here with place New_Tasks in Figure 4.3 . It is of compound color-set type ' Product ' indicating tasks keep their own IDs and deadlines i.e. (New_ID, New_Deadline). e.g. marking $1(6,4)$ means 1 co-efficient of Task ID '6' and deadline '4 time units'
S	1dot	A token at this place is like one-server machine and moves out immediately once all the tasks are completed.

Table 4.5: Description of Set of Places in Figure 4.2

Place	Color_Set	Description
JOB_POOL	T_IDXDeadline	It represents jobs submitted by the user initially. Each token in this place is interpreted as an instance of a job with two fields i.e. unique Task ID and Deadline indicated here as: <i>Colorset T_IDXDeadline = product with TaskId, Deadline.</i>
Task_IN_Ready	T_IDXDeadline	Tasks are arriving after exponential time delay to Ready queue (i.e. tasks are available for execution). Tasks whether pre-empted or suspended ultimately comes to the ready queue.
TaskId	ID	Unique Task IDs are separated to keep track of each task identity
Task_Deadline	Deadline	Deadline of the tasks are isolated in this place that will make it easy to apply deadline based $gEDF$ scheduling policy.
NewTask	Deadline	It maintains the deadlines of new tasks only if it is less than deadline of already submitted tasks depicted here with the arc expression: $([new_DL < DL] 1'new_DL)$ and hence priority is assigned to new task.
Procs_Idle	Dot	It represents the initial available processors. Token at this place indicates that processor is idle. All the processors are homogeneous so default color-set " Dot " is assigned to this place.
RUN	ID	It represents task is being executed by the processor. The priorities are checked before this place so highest priority task will always execute first and processor is now busy.
Waiting	T_IDXDeadline	This place maintains all the waiting tasks along with their IDs and Deadlines. Tasks in this state means all the processors are busy and so ready queue tasks are suspended.
Pre_empt	T_IDXDeadline	It indicates preempted tasks as running tasks are interrupted by high priority task. Preempted Task IDs and their remaining time to deadline (dl) are maintained in this state
TaskExpire	ID	It represents the expired tasks that do not complete their execution within deadline. Terminated tasks are identified through unique IDs. This state should always be empty for successful completion of all the task depicted here with inhibitor arcs that has expressions $1'NewId$ and $1'TaskId$.
Deadlines	Deadline	It represents initial deadlines of Job_Pool tasks that are then used to maintain the clocks and makes sure that no deadline will be missed.
New_DL	Deadline	This place represents Deadlines of newly arrival tasks to keep track of missing deadlines.
RESIDUAL	Deadline	This place holds remaining time to deadline(dl) of each task evaluated through CLOCKS to determine when the running task should be preempted so that there is no starvation of tasks. The output directed arc from place <i>RESIDUAL</i> to transition <i>PREEMPTION</i> is labeled with variable ' dl '.
CompletedTasks	Dot	It indicates task is completed after exponential distributed execution time. A token is added to this place once the task completes its execution within deadline.
Successful	Status	Represents whether all the tasks complete their execution within deadline. This place is of boolean type and evaluates to " true " only when tokens at ' <i>CompletedTasks</i> ' becomes equal to Total tasks depicted here with equal arc.

Table 4.6: Interpretation of Transitions in Figure 4.2

Transition	Guard	Description
START	–	Indicates system is functionally ready to work. This event occurs when jobs are ready to be processed i.e. its pre-place JOB_Pool holds some jobs. The system begins and Task arrives randomly in Task_IN_Ready state with exponential firing rate.
T1	–	It fires once the task arrives to ready queue and splits TaskIDs & Deadlines to place them in their corresponding states i.e. TaskId & Task_Deadline , respectively.
Check_Priorities	–	It checks and sorts the deadlines of ready queue tasks continuously and the least deadline task is given the highest priority. This is done by defining predicates and rate functions corresponding to each deadline (see Table 5.1). Firing of this transition means processor is assigned to task and now added to RUN state for execution.
Check	–	A module shown in Figure 4.3 is a very important macro transition, responsible to compare former deadlines (DL) with new task deadline (new_DL) that may come after random time interval. If the condition ' $New_DL < DL$ ' fulfills then highest priority is given to new task otherwise new task will go to ready queue and assigned some low priority according to algorithm, depicted here with the arc expression $[new_DL \geq DL] 1^*(NewId, new_DL)$ from Check to Task_IN_Ready .
New_Priorities	$NewId <> TaskId$	It examines new deadlines and again check for priorities such that highest priority is given to task with least deadline according to \mathcal{EDF} algorithm. The predicates and rate functions are depicted in Table 5.1.
Execution	–	Running computational tasks are executed with the exponential firing rate and accomplished after elapsing of execution time with a dot added to the post-place CompletedTasks .
Suspend	–	This event occurs when no processor is available and tasks are ready for execution means active tasks have to wait until the processor becomes idle.
PREEMPTION	$new_DL < dl$	It indicates that running task should be preempted. It is fired when the guard is satisfied i.e. if deadline of new task is less than residual deadline of running task.
Resources_Available	–	same as in Table 4.2
CLOCKS	–	The hierarchy of CLOCKS presented in Figure 4.4 is a macro-transition behind that clock for each task is running according to the deadlines in order to make certain that no deadline miss happens and task does not expire.

Chapter 5

Results & Discussions

The stochastic simulation results of \mathcal{SPN} and \mathcal{SPN}^c models (Figure 4.1 and Figure 4.2) are presented in this chapter. Qualitative analysis of some structural properties of the model are determined using \mathcal{PN} analysis tool *Charlie*. Verification results of probabilistic model properties from *PRISM* tool are also presented in this section. Gillespie's Algorithm [87] provided by *Snoopy* tool is adapted in this work for simulations of non-deterministic real time scheduling problem. These stochastic simulations follow Monte-Carlo method that plays a pivotal role to analyze probabilistic nature of complex systems [88].

5.1 General Job Scheduling Simulation Results Using Snoopy

SPN model illustrated in Figure 4.1 is analyzed through stochastic simulations obtained from *Snoopy* tool. The initial markings (I_0) are specified as: $N = 10$, $M = 5$, $JOB_POOL(0) = 10$, $Ready(0) = 0$, $Run(0) = 0$, $Resources(0) = 5$, $Waiting(0) = 0$, $TasksCompleted(0) = 0$ and firing rate of transitions are: $\omega_1 = 0.9$, $\omega_2 = 0.9$, $\omega_3 = 0.02$, $\omega_4 = 0.5$, $\omega_5 = 0.5$, $\omega_6 = 0.1$ and $\omega_7 = 1$.

These parameter values applied to SPN model (Figure 4.1) result in average stochastic simulation graph shown in Figure 5.1. This graph shows dynamic behavior of all the SPN states with respect to simulation time. Initially, all the processors are idle and tasks are in job pool so JOB_POOL and $Resources$ curves are at their maximum. The tasks arrive with exponentially distributed minimum inter-arrival time to the ready queue as the simulation plot depicts. It is observed that the trace of $Ready$ is increasing when JOB_POOL is decreasing in the initial timings indicating Jobs have been arrived and tasks are ready for execution. Over the time as the $Resources$ drop-off, Run trace increases. It implies processor is busy in executing the tasks. It is also noticed from the graph that JOB_POOL is decreasing rapidly in the initial timings and then it slowly rises up since more jobs are continuously added to the JOB_POOL , once initial assigned tasks (N) are completed indicating the system is reversible and live. Moreover, the sinusoidal curve of $Ready$ is slightly higher than Run trace in the beginning then

Run abruptly increases as *Ready* curve approaches to zero. It indicates more tasks are in execution than ready queue. The end behavior shows that *Run* curve lags behind exponentially fast growing *Ready* curve because the process initiates again and in this way jobs are progressing towards completion. The graph also shows that the non-linear *TasksCompleted* trace activates with faster rate but by the time processors become busy, task completion rate slightly slows down since they are suspended.

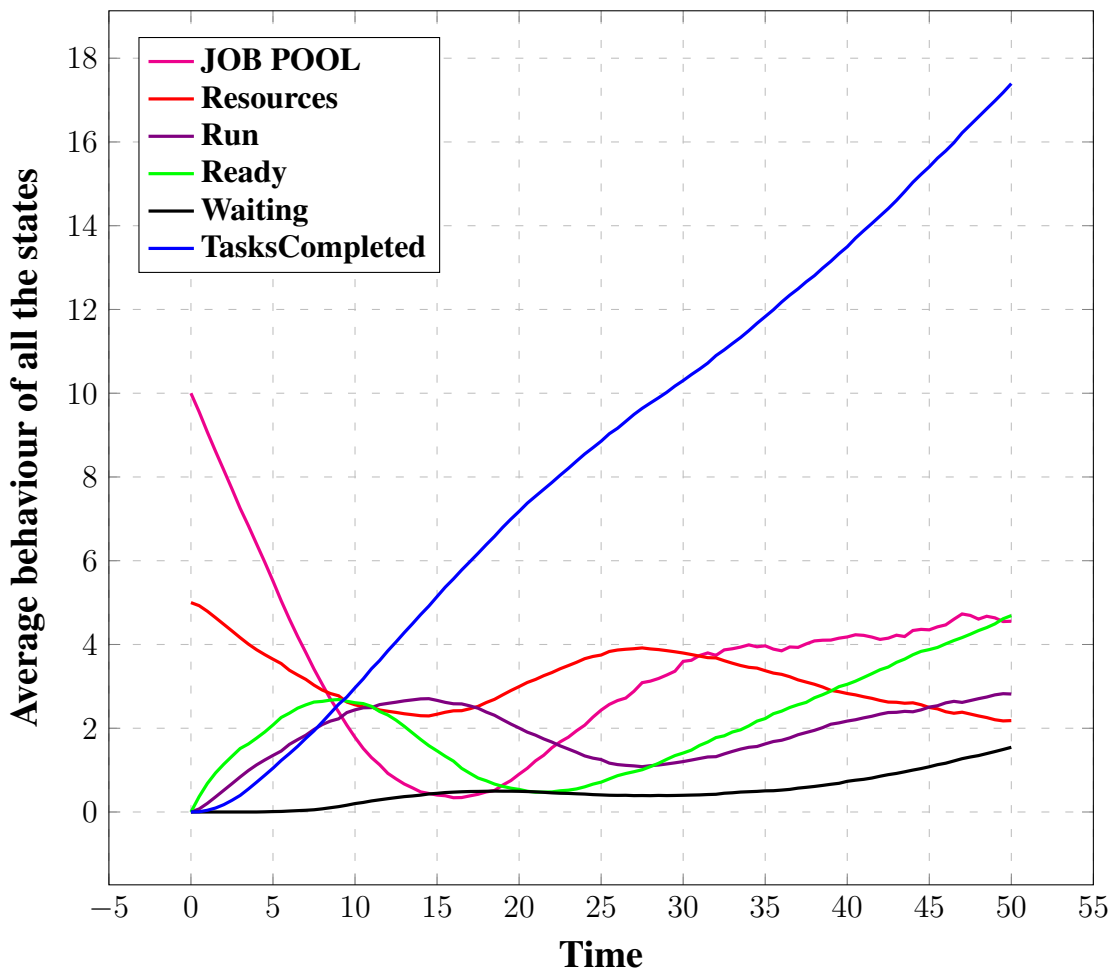


Figure 5.1: Stochastic simulation of General Job Scheduler SPN model presented in Figure 4.1

These behaviors can be analyzed more intelligibly from Figure 5.2 that indicates as the *Resources* decreases gradually in the initial timings, the *Ready* and *Run* traces are rising, however, *Waiting* peak increases when *Resources* approaches to zero and vice versa. Thus, it is inferred from the graph (Figure 5.2) that there is a negative correlation between *Resources* and *Run* trace. Additionally, system performance can be optimized based on certain active resources, ready & waiting time and so forth the study of general job scheduler serves as a basis for advance scheduling schemes.

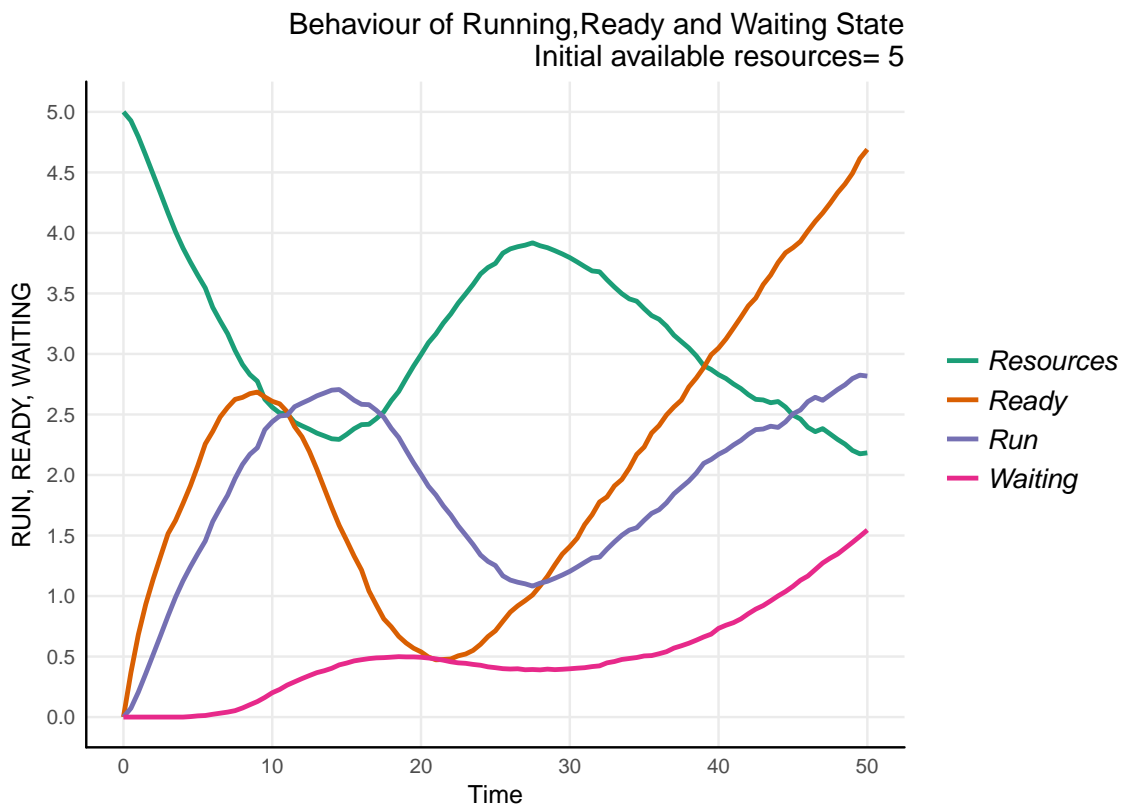


Figure 5.2: Dynamic behavior of significant states in SPN model of General Job Scheduler. Simulation plot showing how the different states in Figure 4.1 affects each other and their evolution with respect to time.

5.2 Simulation Results of SPN^c Model of $gEDF$ Using Snoopy

In this section, the dynamic behavior of $gEDF$ is analyzed and investigated from stochastic simulation results of SPN^c model presented in Figure 4.2. The initial experimental marking values are given in Table 4.4 with the transition firing rates in Table 5.1.

Figure 5.3 shows general behavior of proposed SPN^c model. The stochastic simulation result indicates the evolution of system with respect to time. This scenario considers mapping of 10 deadline-based tasks to four available processors that are idle initially. At 0 time unit, *Procs_Idle* is at its maximum while all other entities are at minimum level showing all processors are free and no task is in execution. From *JOB_POOL* tasks go to *READY* and from there into *RUN* making processors busy. The graph is clearly showing this behavior; *Procs_Idle* curve decreases and *RUN* trace increases which means tasks are being executed by the processors. *Procs_Idle* and *RUN* are affecting *CompletedTasks*, *Waiting* and *Pre_empty* as they tend to increase with increasing *RUN* trace and decreasing *Proc_Idle*. Once *Procs_Idle* approaches its minimum level, *RUN* attains its maximum level indicating all processors are busy being occupied by active tasks, *Waiting* reaches its highest point means new arriving tasks are waiting for processors to get free. *Pre_empty* peak shows highest priority tasks in Ready queue that cause preemption of tasks in execution. Moreover, the behavior

of $Task_{Expire}$ is very significant for scheduling of deadline-based algorithms which must be zero over the period of Job Execution. It is demonstrated from the graph that $Task_{Expire}(t) = 0 \forall t \in [0, \infty)$. Therefore, all the tasks are completed within deadline and no task expires. Finally, when $CompletedTasks$ reaches to maximum, $Successful$ eventually becomes 1 (or true) that shows all the tasks are executed successfully. $CompletedTasks$ rises up to maximum while Run and Waiting curves reach zero which make processors idle all over again. None of task is in waiting constantly and all tasks execute within deadline ($Run = 0$, $Waiting = 0$).

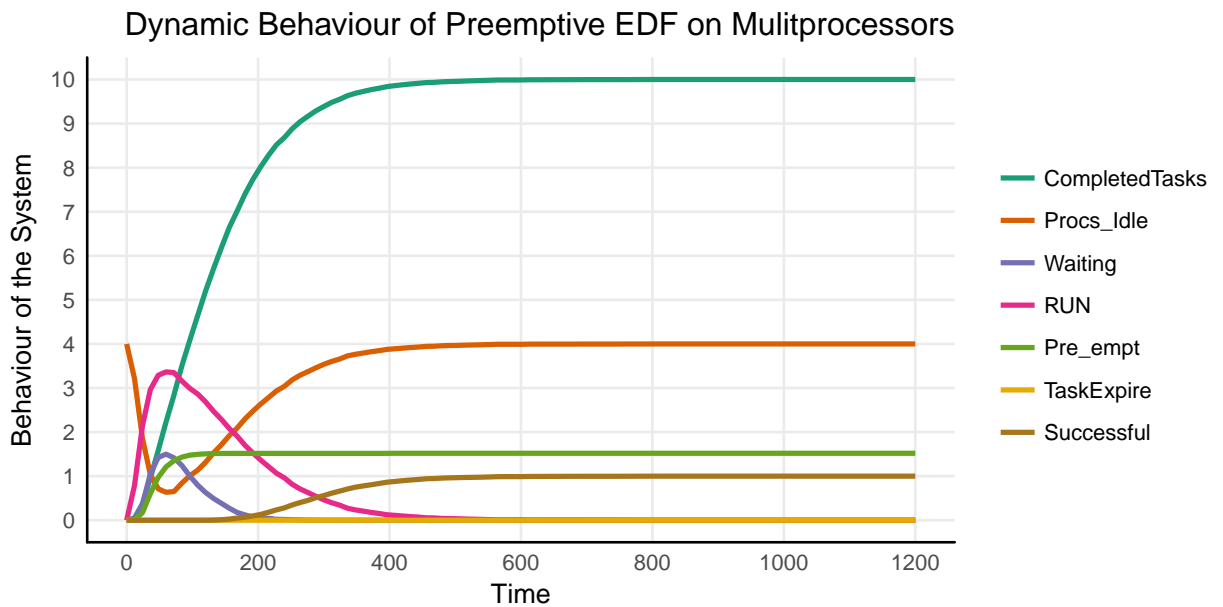


Figure 5.3: Stochastic simulation plot of SPN^c model presented in Figure 4.2 with transition firing rates in Table 5.1 and initial markings listed in Table 4.4

Table 5.1: Rate Functions of transitions in SPN^c model (Figure 4.2)

Transition	Predicate	Rate Function
START	true	0.7
T1	true	0.05
Check_Priorities	DL=1	1/3
	DL=2	1/5
	DL=3	1/15
	DL=4	1/25
	DL=5	1/35
	DL=6	1/45
	DL=7	1/55
	DL=8	1/65
	DL=9	1/75
	DL=10	1/85
T11 (Transition behind ‘Check’)	true	0.5
New_Priorities	new_DL=1	1/10
	new_DL=2	1/20
	new_DL=3	1/30
	new_DL=4	1/40
	new_DL=5	1/50
	new_DL=6	1/60
	new_DL=7	1/70
	new_DL=8	1/80
	new_DL=9	1/90
	new_DL=10	1/100
Execution	true	0.009
Suspend	true	0.8
PREEMPTION	true	0.08
Resources_Available	true	0.05

Figure 5.4 shows the interdependencies of different states in SPN^c on the basis of results obtained in Figure 5.3. The dynamic behavior of *Waiting* and *RUN* traces with respect to *CompletedTasks* is depicted. *Procs_Idle* and *Run* curves are showing negative

correlation and are inversely proportional to each other since processors are occupied by the tasks for execution *Run* curve starts rising. It is also observed from the graph that when upside down curve of *Procs_Idle* reaches to its trough, *Waiting* curve rises up to peak point depicting that new arriving tasks are waiting for the processor to be idle again and *Waiting* starts decreasing as processors are getting idle. In addition to that *Run* is higher than *Waiting* curve that indicates more tasks are in execution than waiting which is the desired behavior. When all the tasks get completed (10 in this case), *Run* and *Waiting* trace converges to zero point while *Procs_Idle* curve has flattened out since all the resources are now available. These significant behaviors will help to optimize the system as well.

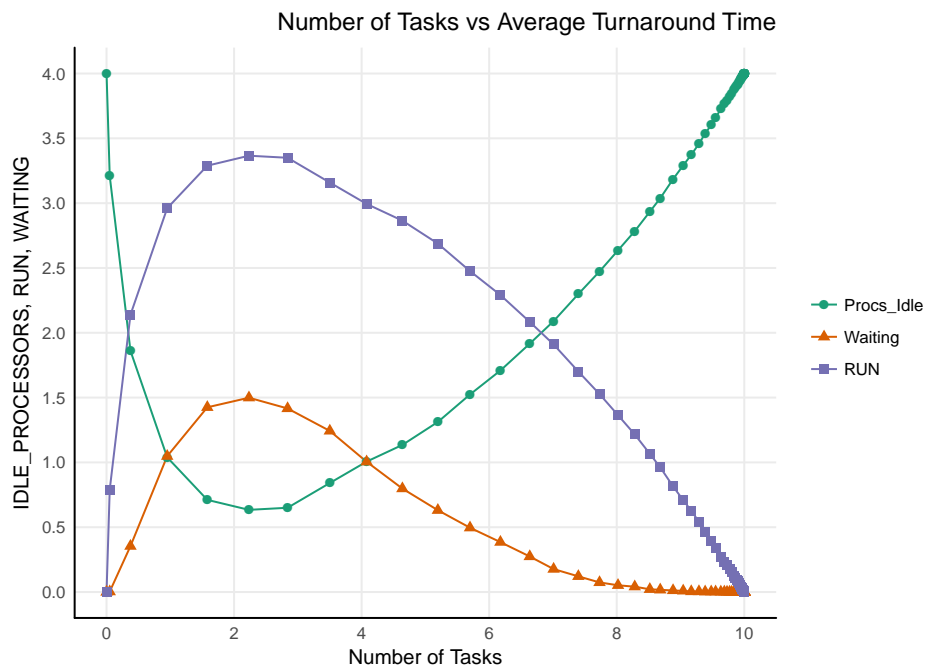


Figure 5.4: Behavior of different states of SPN^c with respect to Tasks Completion. Graph shows mean time from submission of 1st task till the completion of last task. It includes execution time as well as waiting time, also termed as average turnaround time

Furthermore, SPN^c model is analyzed systematically based on different scenarios that will help to evaluate the performance and predictability of $gEDF$ scheduling model as well. Figure 5.5 shows the dynamic behavior of the system when available resources and tasks are equal. The simulation result shows that RUN curve is always lagging behind $Procs_Idle$ and trace of $CompletedTasks$ rises up with high rate because tasks are completing their execution more quickly as compared to Figure 5.3. Also, $Waiting$ trace is constantly zero means no task has to wait and $Procs_Idle$ curve is upside down because the processors are getting busy initially but they are available again after execution of tasks. Finally, $Successful$ curve rises up to 1 abruptly indicating all the tasks accomplished within deadline.

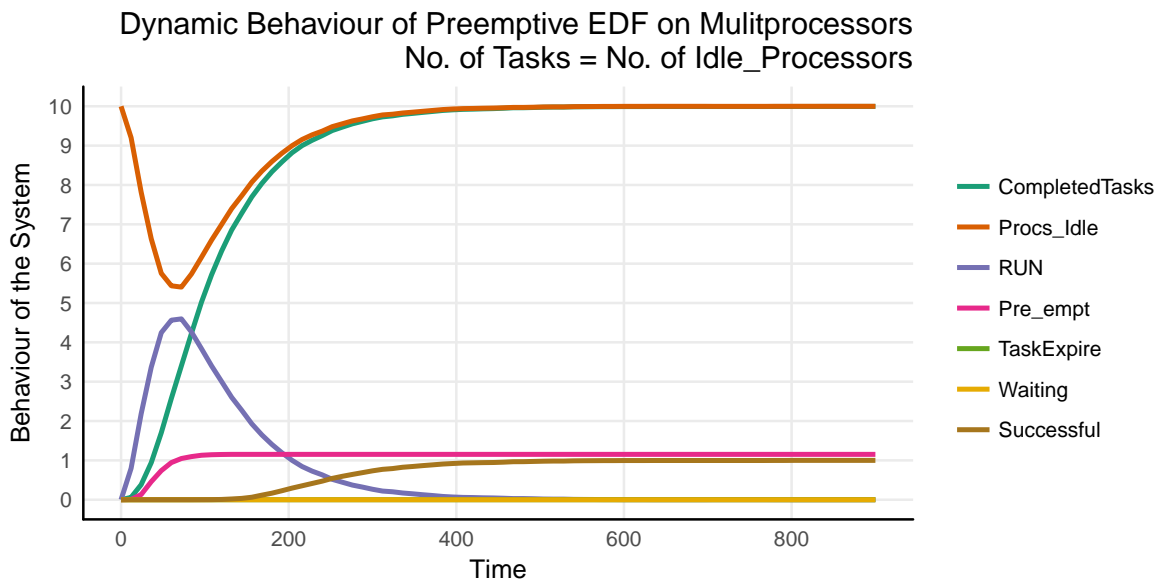


Figure 5.5: Scenario 1. Behavior of stochastic simulations of proposed SPN^c model when available resources are equal to number of tasks. For initial markings, Table 4.4 is modified such that $Procs_Idle=10$ whereas stochastic transition rates are same as in Table 5.1

The graph in Figure 5.6 shows the effect of deadlines on waiting state. In this scenario, deadline varies from 1 to 10 time units with 1 being the highest priority task while 10 the lowest. The initial markings are modified such that $Job_Pool = I(1, N)++ I(2, N)++ I(3, N)++ I(4, N)++ I(5, N)$; $New_Tasks = I(5, N)++ I(6, N)++ I(7, N)++ I(8, N)++ I(9, N)++ I(10, N)$ for the trace of *Deadline N*. The firing rates corresponding to each deadline are listed in Table 5.1. The behavior of $gEDF$ model is analyzed by keeping the same deadline (or same priority) for each task (*JOB_Pool* as well as *New_Task*). All ready tasks with the same deadline have equal priority to execute depicted behind the transitions *Check_Priorities* and *New_Priorities* in Figure 4.2. It is observed from the graph that waiting function shows minimum peak value for the largest deadline trace because tasks are arriving with slower rate. Consequently, new task arrives once the running tasks finish execution or near to completion and processor is free indicating less tasks are in *Waiting* queue. In contrast to that, the curve of smallest deadline trace exhibits maximum peak waiting function because task arrival rate is high. Due to which any new task that arrives will go to waiting since all processors are busy in executing the *RUN* tasks. New tasks reside in waiting state until the processor becomes idle again (note that initial idle processors are less than total tasks). With the passage of time, number of tasks from waiting queue start reducing as they are assigned to processors for execution. Thereupon, all the curves are exponentially decaying and finally converge to zero point. The results obtained in this scenario highly depend on firing rates. Hence, the entirely different behaviors will be obtained by altering the rates

of transitions associated with each deadline (Table 5.1).

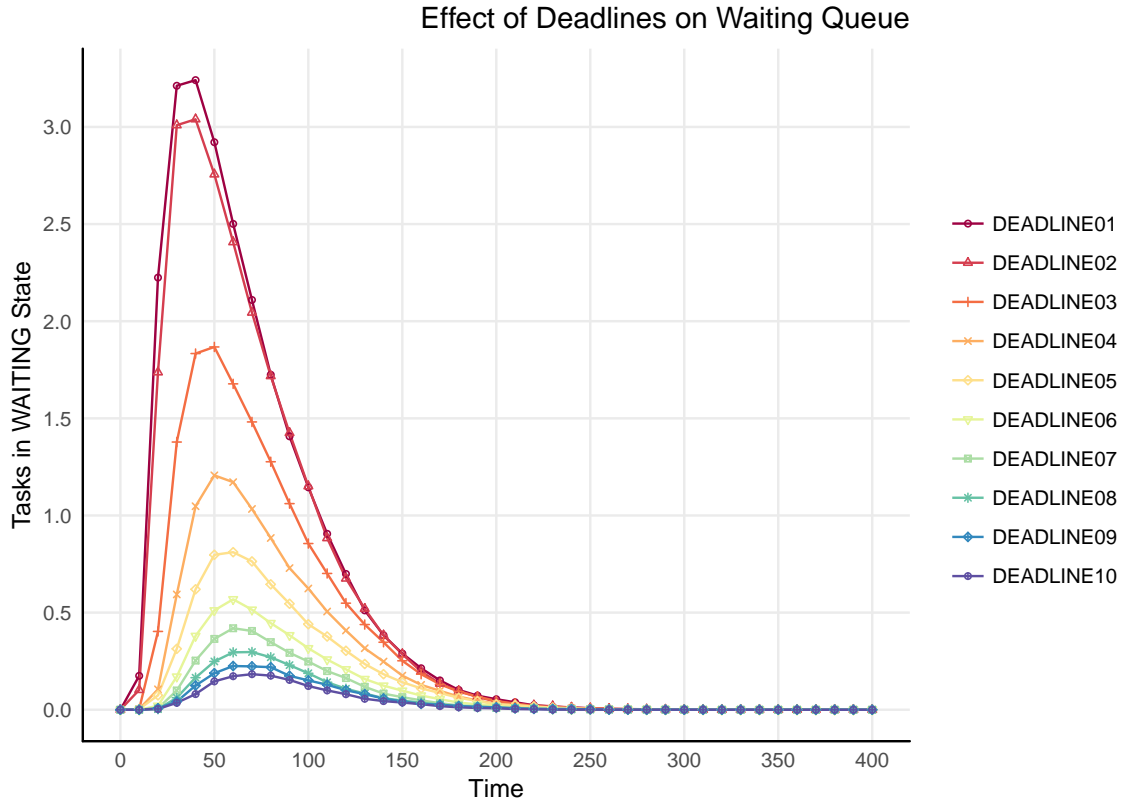


Figure 5.6: Scenario 2. SPN^c model simulation results of Waiting queue for varying deadlines with constant number of tasks and processors. On a time scale from 1 to 10 with 1 being the highest priority task while 10 is lowest. Transition rate functions are listed in Table 5.1. The initial markings are : $Job_Pool = I(1,1)++ I(2,1)++ I(3,1)++ I(4,1)++ I(5,1)$; $Procs_Idle=4\`all()$; $New_Tasks=I(5,1)++ I(6,1)++ I(7,1)++ I(8,1)++ I(9,1)++ I(10,1)$ for the trace of *Deadline01*, it means all the tasks have same priority. Similarly, the simulations are rerun for each trace by varying initial deadlines.

The box plot in Figure 5.7 displays the average throughput of $gEDF$ model with varying number of processors. As the number of multiprocessors increases, rate of completion of average number of tasks also increases. The initial markings of SPN^c in

Table 4.6 are modified such that $Procs_Idle=n\dot{}$, for $Processors_n$ and for all the traces 10 deadline-based tasks are considered. The median line in each box shows the average throughput and possible variations in completed tasks from minimum to maximum values for each processor trace. This also assists in the identification of outliers for different deadline-based tasks in Job_Pool. The number of processors directly affects the throughput of the system. It is also observed that average tasks completion ratio increases as the number of initial idle processors becomes greater than half of total tasks.

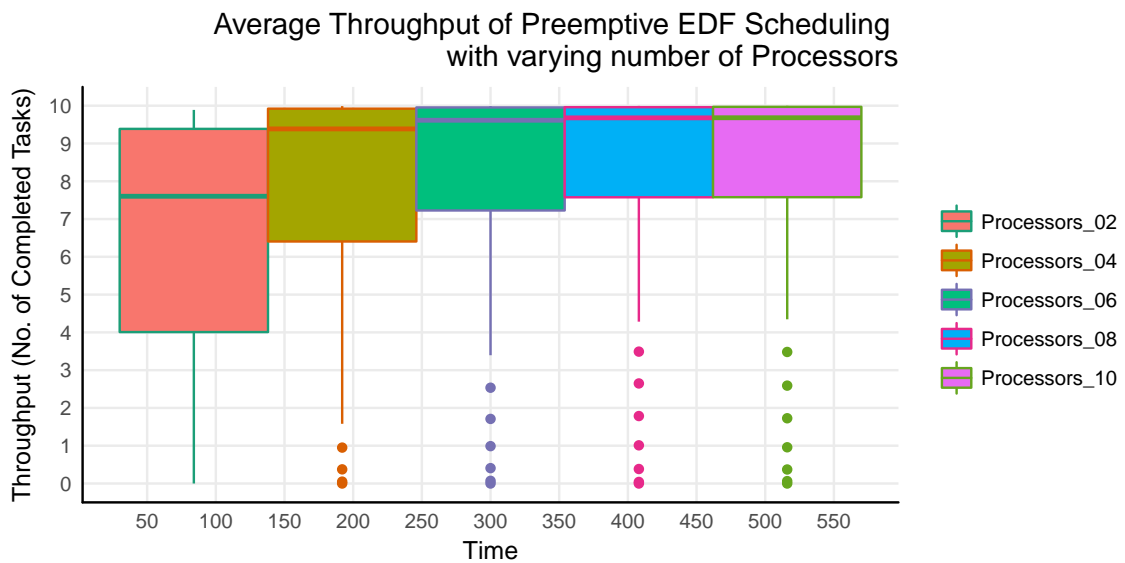


Figure 5.7: Scenario 3 (a). The box plot obtained from simulations of SPN^c model with varying available processors initially (i.e. $Procs_Idle=n\dot{}$, for $Processors_n$) to determine average throughput using $gEDF$ scheduling algorithm according to the idle multiprocessors. Remaining initial markings are same as in Table 4.4 and firing rates for transitions are as in Table 5.1.

In Figure 5.8, throughput (Number of tasks completed per unit of time) is analyzed based on varying initial number of idle processors. The rate functions are listed in Table 4.6 and initial markings given in Table 4.4 are modified such that $Procs_Idle=n\dot{}$. It is observed that multiprocessors greatly affect the performance of scheduling algorithm. The simulation plot shows that throughput curve becomes more steeper as the number of processors increases and it becomes shallow as the number of processors decreases. Processors_N trace is steep if $N \geq \frac{1}{2} * (TotalTasks)$. There is a strong steady growth in throughput when initial available processors increases indicating tasks are completing more quickly. The shallow throughput curve of *Processors_02* trace indicates that it takes long time to complete the tasks.

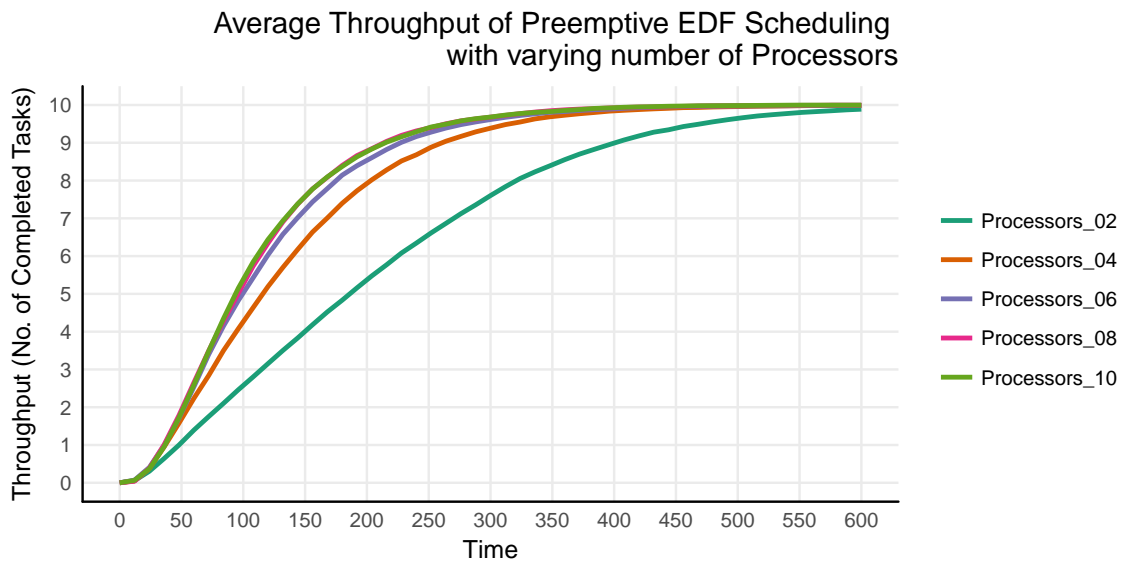


Figure 5.8: Scenario 3 (b). Stochastic simulation plots of SPN^c model to determine throughput by applying $gEDF$ scheduling policy with varying initial multiprocessors. Here total number of tasks are 10. Initial markings given in Table 4.4 are modified such that $Procs_Idle=n\dot{}$, for trace *Processors_n* and transition rate values are listed in Table 5.1.

Figure 5.9 and Figure 5.10 specify that there is a strong relationship between latency, throughput and execution time of tasks. Latency of tasks becomes high as the number of processors are increasing because tasks take long time to run. Figure 5.9 shows that *Processors_N* trace has maximum peak for $N \leq \frac{1}{2} * (TotalTasks)$ because more tasks are in waiting state indicating all processors are busy. When number of processors are equal to tasks, no task has to wait for the processor whereas there is larger area under the curve for *Processors_02* trace for two processors because higher number of tasks are waiting for processors. All the *Waiting* queue curves drop off steadily and finally converges to zero indicating that none of task is in waiting state. As opposed to waiting state, Figure 5.10 shows that greater number of processors accelerates the task completion process. Thus running curves are exponentially growing with the increasing number of processors indicating more tasks are in execution instead of waiting. It also depicts that running queue curve for *Processors_02* trace is short and wider than other traces means tasks are coming to running state slowly and continuously until all the tasks complete execution. *Processors_10* curve grows and drops off sharply indicating tasks are executing very quickly. Moreover, Running queue curves in Figure 5.10 diminish when throughput curves (Figure 5.8) rises up to maximum value. Hence, the algorithm will be more efficient when processors are closed to total number of tasks and in this way highest peak performance is achieved.

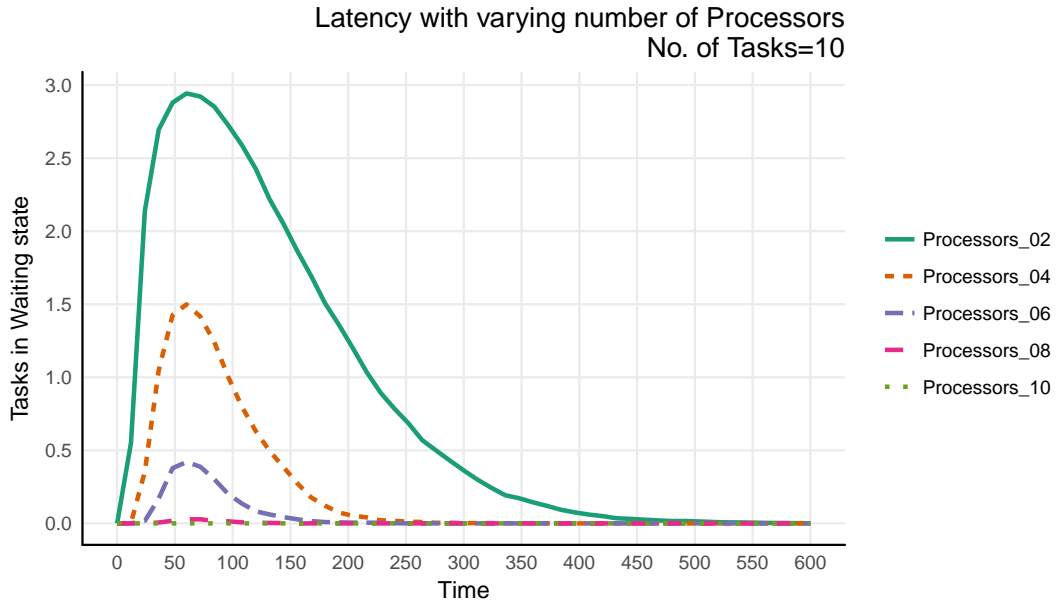


Figure 5.9: Scenario 4. The stochastic simulation behavior of latency for varying number of processors with constant number of tasks applied to the SPN^c model of $gEDF$. Each trace is obtained by modifying initial markings (Table 4.4) such that $Procs_Idle=n\dot{}$, for trace $Processors_n$ and transitions rates are as in Table 5.1

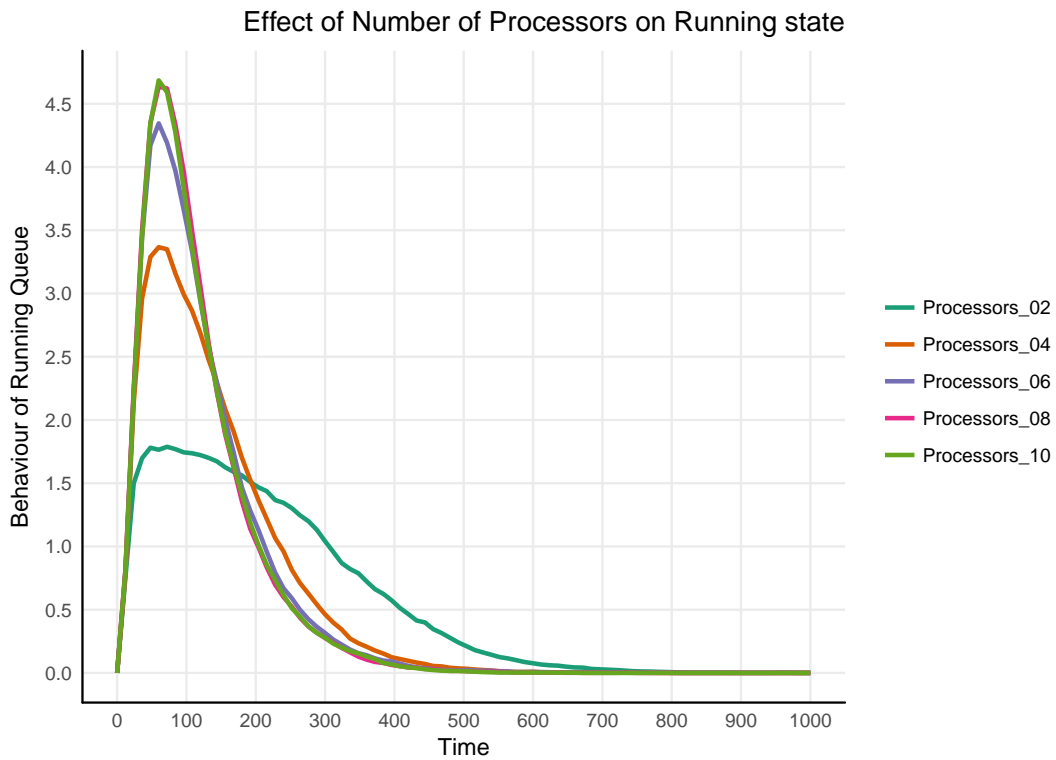


Figure 5.10: Scenario 5. The stochastic simulation results obtained from simulations of SPN^c model with different initial idle multiprocessors to determine their effect on execution of tasks (RUN state in model) by applying $gEDF$ scheduling policy. For initial markings Table 4.4 is modified such that (i.e. $Procs_Idle=n\dot{}$, for $Processors_n$) whereas firing rates for transitions are listed in Table 5.1.

5.3 Verification Results

The behavior of model is exhaustively analyzed using formal model checking approach. Probabilistic model checking is applied to analyze and verify the stochastic behavior of SPN and SPN^c model. This SPN model is transformed into $CTMC$ and the model properties encoded in continuous stochastic logic (CSL) over $CTMC$ model are verified via $PRISM$ tool. $PRISM$ is very powerful and an open-source probabilistic model checking software tool widely used in diverse application areas [84]. In CSL , ‘ P ’ the probability operator replaces PCTL path quantifiers ‘ A (forall)’ and ‘ E (there exists)’. It takes the form as: $P \text{ bound } [\text{pathprop}]$ where pathprop is the path property to be verified and path from state s must meets the probability bound which has range $[0,1]$. The $PRISM$ properties are customized using filters. A keyword filter is used to specify the filters that verify the properties simultaneously from all the states of the model, formulated as: $\text{filter}(\text{op}, \text{prop})$, where op is the operator applied to the values of prop probabilistic property and returns single value [84]. The model parameters for experimental purposes are defined as: $Max = 10, N(0) = 1, M(0) = 1, T_{\omega1} = 0.9, T_{\omega2} = 0.8, T_{\omega3} = 0.02, T_{\omega4} = 0.5, T_{\omega5} = 0.5, T_{\omega6} = 0.1, T_{\omega7} = 1$. Various queries in CSL language are defined to deeply investigate the behavior of the model and its verification.

The specification properties are given and discussed as follow:

- **Deadlock-Freeness Property:** The deadlocks in the model are inspected by using PRISM filter queries that test the existence of deadlock state from all the reachable states. Deadlocks can also be identified by using the *print* operator.

$$\text{filter (exists, ! "deadlock")} \text{ or filter (print, "deadlock")} \quad (5.1)$$

The property result of *filter (exists, ! "deadlock")* is *True* that verifies the system is deadlock-free. A real time scheduler should never be in a state where no progress is possible. Hence, our model satisfies the desirable feature of the real time systems.

- **Safety Property:** Safety property is defined as something bad must never happen during execution of the system under provided circumstances [89]. It is formally presented for the model as:

$$P \leq 0 [G(\text{TasksCompleted!} = N \& \text{Resources} \geq 1 \& \text{Run} = 0 \& \text{Waiting!} = 0)]$$

$$\text{or } \text{TasksCompleted!} = N \& \text{Resources} \geq 1 \Rightarrow P \geq 1 [GF(\text{Run!} = 0)] \quad (5.2)$$

This property is stated as "the probability that tasks are in running state when processors are idle is always zero". It is verified in our model means when *TasksCompleted* is less than *TotalTasks* and processors are idle then tasks must be in running state instead of waiting state.

- **Fairness Property :** Fairness property means a desirable event of the system is occurring frequently, formally expressed for our model as:

$$Waiting = 1 \Rightarrow P \geq 1[G(X(Run = 1))] \quad (5.3)$$

This property asserts that if the task is in *waiting* state then next state will always be the run state. This event will occur infinitely often.

- $P =? [G \text{ Resources} = 0]$: This property inquires about processors utilization. It evaluates to '0' that indicates processors are always busy in executing the tasks. Thus, it shows maximum processors utilization and minimum response time.
- $\text{filter}(\text{forall}, P \geq 1 [F(\text{Waiting} = 0) \& (\text{Run} = 0)])$: All the waiting and running tasks will eventually be completed successfully. It is formalized using *filters* with *pathop* 'forall' which checks that starting from any reachable state, *Run* and *Waiting* states will eventually reach to zero with probability 1. This query is verified in our model and hence follows the simulation results.
- $(P \geq 1 [G((\text{Waiting} \neq 0) \cup F(\text{Resources} \geq 1))])$: The assertion query is stated as, "The probability that, globally, tasks will be in waiting state until processors become available again is 1". This global property evaluates to true and thus verifies the simulation results.
- $S =? [(\text{Run} > \text{Waiting})]$: This query inquires, "What is the steady-state probability of tasks being more in *running* state than *Waiting* state?". The result shows, in long run there is 73% probability that tasks in execution are greater than waiting. Thus, it proves that waiting time is reduced.

- $S = ?[(Resources < Max)]$: This property is stated as, “What is the long run probability that *Resources* are less than maximum available processors?”. This query evaluates to “0.999” indicating most of the time processors are busy in executing the tasks. This property is formulated in another way as:

$R\{“time”\} = ? [F Resources = Max]$ which means “What is the expected time that resources reaches to Maximum value?”. The expected reward is *infinity* in this case, which indicates a state, where all the processors are idle, will never be reachable.

A *unit* reward to each model state is assigned using reward structure “*time*” and different accumulated reward properties are determined.

- $R\{“time”\} = ? [F Ready > 0]$: This property determines the expected time of the task to reach the ready state. The result is 0.1111 time units.
- $R\{“time”\} = ? [F Run > 0]$: It inquires, “What is the expected accumulated reward for the tasks to eventually reaches to *Run* state?”. The expected reward is 0.5494 time units from the initial state.
- $R\{“time”\} = ? [F Waiting > 0]$: This query interrogates, “What is the expected accumulated reward time for the tasks to eventually reaches to *Waiting* state?”. The expected reward result is 36.2899 time units from the initial state. The above accumulated reward results indicates that tasks in waiting state are always less than active and executing tasks.

- $(P \geq 1 [F((JOB_POOL = 0) \& (TasksCompleted = N))])$: This assertion query is stated as, "From the initial state, eventually JOB_POOL becomes empty and all the tasks are completed with unit probability ". This property holds in our model which proves that all the scheduled tasks are executed successfully.

5.3.1 Stochastic Model Checking of SPN^c Model of $gEDF$

In order to explore the behavior of SPN^c (Figure 4.2), the first step is to unfold SPN^c into corresponding flat SPN and then investigate its $CTMC$ in *PRISM*. For spatial limitations, $CTMC$ of the model is not presented here. The following CSL queries are formulated for probabilistic validation of $gEDF$ model:

- **Query 1:** $P \geq 1 [X \text{ true}]$. It is deadlock freeness property which states, "The probability that each state has the next state is 1". This property is satisfied in our model thus guarantees that there are no deadlocks in the model.
- **Query 2:** $P =? [G \text{ Procs_Idle_dot} = 0]$. This property inquires, "What is the probability that idle processors become zero ". It evaluates to '0' which implies processors become free and available again after execution. Most of the time processors are busy in executing the tasks, formally stated as, $S =? [Procs_Idle_dot \geq Max]$ and evaluates to '0'. It means maximum processor utilization is achieved as tasks are being occupied by the processors.

- **Query 3:** filter (forall, $P \geq 1 [F ((\text{RUN}_N = 0))]$), where $N = \{ 1, 2, \dots, \text{Total_Tasks} \}$: All the running tasks must be executed eventually. The assertion query is formalized using *filters* in PRISM which states that the probability to reach a state where RUN becomes zero is unity. This property is verified in our model.
- **Query 4:** $P =? [F (\text{CompletedTasks_dot} = \text{Max}) \& (\text{Successful_true} = 1)]$. This query interrogates, “What is the probability that completed tasks reach to maximum and the status of successful becomes high?”. It is verified with probability ‘1’ in our model, hence signifies an important **“schedulability property”** that all the tasks are completed successfully.
- **Query 5:** $P \geq 1 [G ((\text{TaskExpire}_N = 0))]$. It is the **“safety property”** of our model that states, “System will never reach the state where task expires”. This assertion query is verified in our model, thus implies all the tasks are completed within time constraints

The aim of probabilistic model checking is to determine the verification of various attributes of task scheduling and to identify the failures. The two most important attributes of real time scheduling algorithms are “schedulability” and “correctness” [33], presented above in *Query 4* and *Query 5*. The proof of these properties signifies that all the tasks are scheduled according to $g\mathcal{EDF}$ algorithm and completed meeting their deadlines. This provides sound knowledge of dynamics of the system.

The structural Petri net properties of \mathcal{SPN} model are determined using qualitative analysis tool *Charlie* [83]. It proves the following behavioral properties of general job scheduling \mathcal{SPN} model in Figure 4.1 :

- **Liveness** : It means each of the transition in \mathcal{SPN} model is enabled for every reachable marking. All transitions in the net evolves infinitely often. It is the most significant property of Petri net model.
- **Reversible** : \mathcal{SPN} model is also reversible. Once the tasks in Job_Pool are completed, the process reinitializes.
- **Deadlock-free**: None of the transitions in the model become disabled on permanent basis. It means there is no deadlock in the system design.
- **Strongly connected**: A live PN must be strongly connected [90]. Our analysis also shows that the net is strongly connected as there exists a directed path between every node.

The results presented above indicates that system is always live and always able to evolve under provided circumstances. Various qualitative and quantitative stochastic specifications are formalized and investigated. The verification of safety properties proves that no task expires and all the tasks meet their deadlines. Fairness property specifies that the scheduling of tasks is performed infinitely often. Deadlock-freeness property indicates there are no deadlocks in the systems. Quantitative properties facilitate us to explore the different performance parameters. The system never achieved

any deadlock nor halted under any condition. This schedules all the tasks successfully before/at deadline, thus, feasibility is also achieved. Moreover, the model verification results are highly coherent with the simulation results. Thus, model checking allows us to evaluate the performance of model and let us ascertain that the system is reliable and valid at each time step.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

In this study, a novel approach was proposed for the probabilistic analysis of dynamic real time scheduling algorithm. An automated colored stochastic Petri net (SPN^c) modeling and probabilistic verification based framework was proposed for the dynamic analysis of real time scheduling algorithms. The model of the system was analyzed via simulations that helped to evaluate different performance parameters efficiently e.g. the throughput, latency and execution rate of the jobs etc.. The model properties were also validated further with stochastic model checking approach e.g. *scheduling property* and *correctness property* were also verified which guarantees that all the tasks were successfully completed.

Real time pervasive multiprocessor systems are widely promoted to life critical applications that must be utterly reliable. The trend to use embedded and ubiquitous computing technology in numerous real time fields is growing which demands a very reliable system without any user interference. The non-deterministic events must be managed such that the system satisfied the rigorous timing constraints. A number of accidents reported in past due to mismanagement of real time tasks e.g. in flying automobiles, Google Self Driving Car etc.. These real time applications demand high computational processing and are highly critical. Reliability and predictability in real time scheduler are interpreted as the extreme likelihood of meeting the critical deadlines of real time tasks. Petri net modeling formalism was applied to analyze the behavior of scheduler. SPN model of general job scheduler was analyzed based on stochastic simulations and model verification. In addition to that, Colored Petri net modeling formalism was utilized to accurately and precisely investigate the non-deterministic behavior of real time scheduling algorithm. A preemptive dynamic priority $gEDF$ deadline-based real time scheduling algorithm for multiprocessor systems was designed using state-of-the-art SPN^c formal modeling approach. In order to explore and evaluate the performance of $gEDF$, the stochastic simulations of SPN^c model are analyzed. Moreover, $CTMC$ of the model was utilized to further examine and validate the probabilistic behavior of the system by the verification of desirable CSL encoded properties. In addition to strategic performance analysis, it was proved in this model that system is deadlock free, live and all the real time tasks meet their deadlines. The probabilistic approaches are not well

exploited earlier for real time systems. Hence, this study will serve as a foundation for specification of real time scheduling algorithms using high level Petri net techniques.

A framework based on automated stochastic modeling and verification will help the designers to deeply analyze the system failures in early design phases and optimize its performance. Hence, it provides guaranteed, mountable and robust solution for dynamic and stochastic real time scheduling problems especially in life-critical applications. The increase in model size directly increases the variables of the system which may result in state space explosion. Accordingly, this study is restricted due to state space explosion problem for very large models and thus the only limitation of this study for the model checking process.

6.2 Future Work

Probabilistic formal methods based methodology proposed in this work has been proved to be persistently reliable. The significance of this study is manifold. This work can be extended in several research areas which must be investigated. One of the extension of this work can be applied to multi-core computing systems for scheduling of multi-threaded parallel task model [39, 91]. Fault tolerance and overload management, two very important attributes in real time scheduling [92, 93], will be encountered when applying formal methods on real time systems. Scheduling of mixed-critical and cyber-physical systems can be done in future, as current study focuses on HRTS.

In later studies, heterogeneous computing platforms as well as partitioned multiprocessor scheduling algorithms will be taken into account. Moreover, real time applications are also security critical [94], so this essential aspect can also be well-explored using probabilistic formal methods.

Bibliography

- [1] Jonathan S. Ostroff. Formal methods for the specification and design of real-time safety critical systems. *Journal of Systems and Software*, 18(1):33–60, 1992.
- [2] J. P. Bowen and M. G. Hinchey. Ten commandments of formal methods... ten years later. *Computer*, 39(1):40–48, Jan 2006.
- [3] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.
- [4] Ernst-Rüdiger Olderog and Henning Dierks. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.
- [5] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1997.

-
- [12] Md Tawhid Bin Waez, Juergen Dingel, and Karen Rudie. A survey of timed automata for the development of real-time systems. *Computer Science Review*, 9:1 – 26, 2013.
- [13] Maissa Elleuch, Osman Hasan, Sofiène Tahar, and Mohamed Abid. Formal probabilistic analysis of detection properties in wireless sensor networks. *Formal Aspects of Computing*, 27(1):79–102, Jan 2015.
- [14] J. Carlier, Ph. Chretienne, and C. Girault. *Modelling scheduling problems with Timed Petri Nets*, pages 62–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.
- [15] Martin Naedele. An approach to modeling and evaluation of functional and timing specifications of real-time systems. *Journal of Systems and Software*, 57(2):155 – 174, 2001.
- [16] W. M. P. van der Aalst. Petri net based scheduling. *Operations-Research-Spektrum*, 18(4):219–229, Dec 1996.
- [17] S. Balaji, L.M. Patnaik, Lawrence Jenkins, and P.S. Goel. S-nets: A Petri net based model for performance evaluation of real-time scheduling algorithms. *Journal of Parallel and Distributed Computing*, 15(3):225 – 237, 1992.
- [18] Samir M. Koriem. R-nets for the performance evaluation of hard real-time systems. *Journal of Systems and Software*, 46(1):41 – 58, 1999.

-
- [19] Emmanuel Grolleau and Annie Choquet-Geniet. *Off-Line Computation of Real-Time Schedules by Means of Petri nets*, pages 309–316. Springer US, Boston, MA, 2000.
- [20] Olivier Henri Roux and Anne-Marie Déplanche. A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation*, 36(7):973–987, 2002.
- [21] Bernard Berthomieu and Michel Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, March 1991.
- [22] W. Karamti, A. Mahfoudhi, and Y. H. Kacem. Using dynamic priority time Petri nets for scheduling analysis via earliest deadline first policy. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 332–339, July 2012.
- [23] Georgeta Igna. *Performance analysis of real-time task systems using Timed Automata*, Doctoral thesis. 2013.
- [24] Victor A. Braberman and Miguel Felder. *Verification of Real-Time Designs: Combining Scheduling Theory with Automatic Formal Verification*, pages 494–510. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

- [25] Y. Abdeddaim, A. Kerbaa, and O. Maler. Task graph scheduling using timed automata. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 8 pp.–, April 2003.
- [26] K. Altisen, G. Gössler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, RTSS '99, pages 154–, Washington, DC, USA, 1999. IEEE Computer Society.
- [27] A. Z. Olivera Salmon, P. M. Gonzalez del Foyo, and J. R. Silva. Scheduling real-time systems with periodic tasks using a model-checking approach. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 73–78, July 2014.
- [28] Rajeev Alur, Costas Courcoubetis, and David Dill. *Model-checking for probabilistic real-time systems*, pages 115–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.
- [29] Didier Lime and Olivier (H.). Roux. Formal verification of real-time systems with preemptive scheduling. *Real-Time Systems*, 41(2):118–151, Feb 2009.
- [30] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: A flexible real time scheduling framework. In *Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for*

- Real-time & Distributed Systems Using Ada and Related Technologies*, SIGAda '04, pages 1–8, New York, NY, USA, 2004. ACM.
- [31] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 298–302. Springer, 1998.
- [32] Walid Karamti and Adel Mahfoudhi. Scheduling analysis based on model checking for multiprocessor real-time systems. *The Journal of Supercomputing*, 68(3):1604–1629, Jun 2014.
- [33] J. Liu, M. Zhou, X. Song, M. Gu, and J. Sun. Formal Modeling and Verification of a Rate-Monotonic Scheduling Implementation with Real-Time Maude. *IEEE Transactions on Industrial Electronics*, 64(4):3239–3249, April 2017.
- [34] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition, 2011.
- [35] Marta Kwiatkowska, Gethin Norman, and David Parker. *Probabilistic Model Checking: Advances and Applications*, pages 73–121. Springer International Publishing, Cham, 2017.

- [36] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multi-processor systems. *ACM Computing Surveys (CSUR)*, 43(4):35:1–35:44, October 2011.
- [37] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2):101–155, Nov 2004.
- [38] Chung Laung Liu and James W Layland. Scheduling algorithms for multi-programming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
- [39] Robert I. Davis. Guest editorial: special issue on mixed-criticality, multi-core, and micro-kernels. *Real-Time Systems*, 53(5):669–672, Sep 2017.
- [40] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 63–72, 2012.
- [41] Sanjoy K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, Jan 2003.
- [42] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling

- problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.
- [43] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, pages 193–202, Dec 2001.
- [44] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.
- [45] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Springer Publishing Company, Incorporated, 2013.
- [46] Uwe Schwiegelshohn and Ramin Yahyapour. Fairness in parallel job scheduling. *Journal of Scheduling*, 3(5):297–320, 2000.
- [47] Jonathan Weinberg. Job scheduling on parallel systems. *University of California, San Diego*, pages 1–12, 2002.
- [48] Theodore P Baker and Michele Cirinei. A unified analysis of global edf and fixed-task-priority schedulability of sporadic task systems on multiprocessors. *Journal of Embedded Computing*, 4(2):55–69, 2010.
- [49] Arezou Mohammadi and Selim G Akl. Scheduling algorithms for real-time systems. *School of Computing Queens University, Technical Report*, 2005.

- [50] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, April 2009.
- [51] Joseph Sifakis. Modeling real-time systems-challenges and work directions. In *EMSOFT*, volume 1, pages 373–389. Springer, 2001.
- [52] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
- [53] Manuel Silva. Half a century after Carl Adam Petri’s Ph.D. thesis: A perspective on the field. *Annual Reviews in Control*, 37(2):191 – 219, 2013.
- [54] René David and Hassane Alla. *Discrete, continuous, and hybrid Petri nets*. Springer Science & Business Media, 2010.
- [55] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.
- [56] M Marsan. Stochastic Petri nets: an elementary introduction. *Advances in Petri nets 1989*, pages 1–29, 1990.
- [57] Elvio Gilberto Amparore, Susanna Donatelli, and Elisa Landini. *Modelling and Evaluation of a Control Room Application*, pages 243–263. Springer International Publishing, Cham, 2017.

- [58] Fernando J Jaimes-Romero, David Muñoz-Rodríguez, Carlos Molina, and Hazem Tawfik. Modeling resource management in cellular systems using Petri nets. *IEEE Transactions on Vehicular technology*, 46(2):298–312, 1997.
- [59] Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. *Modelling with generalized stochastic Petri nets*. John Wiley & Sons, Inc., 1994.
- [60] M. K. Molloy. Performance Analysis Using Stochastic Petri Nets. *IEEE Transactions on Computers*, 31(9):913–917, September 1982.
- [61] Yadati Narahari and N Viswanadham. Performance modelling of a fault-tolerant real-time multiprocessor using stochastic Petri nets. *Sadhana*, 11(1-2):187–208, 1987.
- [62] Hoon Choi, Vidyadhar G. Kulkarni, and Kishor S. Trivedi. Markov regenerative stochastic Petri nets. *Performance Evaluation*, 20(1):337 – 357, 1994.
- [63] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, 2(2):93–122, 1984.
- [64] Kurt Jensen. Coloured Petri nets. In *Discrete Event Systems: A New Challenge for Intelligent Control Systems*, *IEE Colloquium on*, pages 5–1. IET, 1993.

- [65] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 2013.
- [66] Fei Liu, MONIKA Heiner, and CHRISTIAN Rohr. The manual for colored Petri nets in Snoopy– $QPN^C / SPN^C / CPN^C / GHPN^C$. 2012.
- [67] Fei Liu and Monika Heiner. Colored Petri nets to model and simulate biological systems. In *Proceedings of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2010)*, volume 827, pages 71–85. Citeseer, 2010.
- [68] Fei Liu and Monika Heiner. Modeling membrane systems using colored stochastic Petri nets. *Natural Computing*, 12(4):617–629, 2013.
- [69] Christel Baier, Boudewijn R Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Performance evaluation and model checking join forces. *Communications of the ACM*, 53(9):76–85, 2010.
- [70] Junaid Qadir and Osman Hasan. Applying formal methods to networking: Theory, techniques, and applications. *IEEE Communications Surveys & Tutorials*, 17(1):256–291, 2015.
- [71] Edmund M Clarke. The birth of model checking. In *In: Grumberg O., Veith H. (eds) 25 Years of Model Checking. Lecture Notes in Computer Science, vol 5000.*, pages 1–26. Springer, Berlin, Heidelberg, 2008.

- [72] Edmund M Clarke, E Allen Emerson, and Joseph Sifakis. Model checking: algorithmic verification and debugging. *Communications of the ACM*, 52(11):74–84, 2009.
- [73] Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.
- [74] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.
- [75] Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with prism: A hybrid approach. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 52–66. Springer, 2002.
- [76] S. Donatelli, S. Haddad, and J. Sproston. Model checking timed and stochastic properties with CSL^{TA} . *IEEE Transactions on Software Engineering*, 35(2):224–240, March 2009.
- [77] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, Jul 1999.

- [78] Marta Kwiatkowska, Gethin Norman, and David Parker. *Stochastic Model Checking*, pages 220–270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
- [79] Marta Kwiatkowska, Gethin Norman, and David Parker. Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science*, 153(2):5 – 31, 2006. Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005).
- [80] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-Checking Continuous-time Markov Chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, July 2000.
- [81] Monika Heiner, Mostafa Herajy, Fei Liu, Christian Rohr, and Martin Schwarick. Snoopy—a unifying Petri net tool. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 398–407. Springer, 2012.
- [82] Andreas Franzke. Charlie 2.0—a multi-threaded Petri net analyzer. *Brandenburg University of Technology Cottbus*, 2009.
- [83] M Heiner, M Schwarick, and J Wegener. Charlie — an extensible Petri net analysis tool. In R Devillers and A Valmari, editors, *Proc. PETRI NETS 2015*, volume 9115 of *LNCS*, pages 200–211. Springer, June 2015.
- [84] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc.*

- 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- [85] Su-Hui Chiang and Sangsuree Vasupongayya. Design and potential performance of goal-oriented job scheduling policies for parallel computer workloads. *IEEE Transactions on Parallel and distributed systems*, 19(12):1642–1656, 2008.
- [86] Brinkley Sprunt, Lui Sha, and John Lehoczky. Scheduling sporadic and aperiodic events in a hard real-time system. Technical report, Carnegie-Mellon Univ., Pittsburgh, PA (USA). Software Engineering Center, 1989.
- [87] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
- [88] Dirk P. Kroese, Tim Brereton, Thomas Taimre, and Zdravko I. Botev. Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):386–392, 2014.
- [89] Ekkart Kindler. Safety and liveness properties: A survey. *Bulletin of the European Association for Theoretical Computer Science*, 53:268–272, 1994.
- [90] Richard Shell. *Handbook of industrial automation*. CRC Press, 2000.
- [91] H. S. Chwa, J. Lee, J. Lee, K. M. Phan, A. Easwaran, and I. Shin. Global edf schedulability analysis for parallel tasks on multi-core platforms. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1331–1345, May 2017.

-
- [92] C. M. Krishna. Fault-tolerant scheduling in homogeneous real-time systems. *ACM Computing Surveys (CSUR)*, 46(4):48:1–48:34, March 2014.
- [93] G. Manimaran and C. S. R. Murthy. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1137–1152, Nov 1998.
- [94] T. Xie and X. Qin. Scheduling security-critical real-time applications on clusters. *IEEE Transactions on Computers*, 55(7):864–879, July 2006.

Appendix A

CTMC of *SPN* model of General Job Scheduling

```
ctmc //type of probabilistic model

// constants

const int Max; //upper limit

const int N; //Total Jobs

const int M; // idle processors

const double T_omega1; //the rate at which the task arrives

const double T_omega2; //the rate of task scheduling

const double T_omega3; //the rate at which the task may preempt

const double T_omega4; //the rate at which the task is executed

const double T_omega5; //Waiting rate of tasks
```



```
const double T_omega6;//the rate at which
    //the resources become available once task is suspended
const double T_omega7;//the rate at which the process restarts

module GeneralScheduling //Name of the module

//Definition of State Variables
JOB_POOL: [ 0..Max ] init N;
Waiting: [ 0..Max ] init 0;
Ready: [ 0..Max ] init 0;
TasksCompleted: [ 0..Max ] init 0;
Resources: [ 0..Max ] init M;
Run: [ 0..Max ] init 0;

//Definition of Transition rules
//State variables are updated with the corresponding rates
//only if the guard condition fulfills
[Task_Arrives]
(JOB_POOL > 0) & (Ready < Max )-> T_omega1 * JOB_POOL :
(JOB_POOL' = JOB_POOL-1) & (Ready' = Ready+1);
```

[Scheduling]

(Ready > 0) & (Resources > 0) & (Run < Max) -> T_omega2 * Ready:

(Ready' = Ready-1) & (Resources' = Resources-1) & (Run' = Run+1);

[Preemption]

(Run > 0) & (Ready < Max) & (Resources < Max) -> T_omega3 * Run:

(Ready' = Ready+1) & (Resources' = Resources+1) & (Run' = Run-1);

[Ei]

(Run > 0) & (Resources < Max) & (TasksCompleted < Max) ->

T_omega4 * Run : (Resources' = Resources+1) & (Run' = Run-1) &

(TasksCompleted' = TasksCompleted+1);

[Suspend]

(Resources < 1) & (Ready > 0) & (Waiting < Max) -> T_omega5:

(Ready' = Ready-1) & (Waiting' = Waiting+1);

[Resources_Available]

(Waiting > 0) & (Ready < Max) -> T_omega6 * Waiting :

(Ready' = Ready+1) & (Waiting' = Waiting-1);

```
[RESTART]

(TasksCompleted > 9) & (JOB_POOL < 1) ->

T_omega7 * TasksCompleted :

(JOB_POOL' = JOB_POOL+10)&(TasksCompleted' = TasksCompleted-10);

endmodule//end of module

//Definition of the reward

rewards "time"

true : 1; //assign reward 1 to each state

endrewards

// end of reward
```