Modeling Software Systems with Rejuvenation, Restoration and Checkpointing through Fluid Stochastic Petri Nets

A. Bobbio[1], S. Garg[2], M. Gribaudo[3], A. Horváth[4], M. Sereno[3], M. Telek[4]

[1]Dipartimento di Scienze e Tecnologie Avanzate, Università del Piemonte Orientale, Alessandria, Italy

[2]Lucent Technologies, Bell Laboratories, Murray Hill, NJ, USA

[3]Dipartimento di Informatica, Università di Torino, Italy

[4] Híradástechnikai Tanszék, Budapesti Műszaki Egyetem, Budapest, Hungary

Abstract

In this paper, we present a Fluid Stochastic Petri Net (FSPN) based model which captures the behavior of aging software systems with checkpointing, rejuvenation and self-restoration, three well known techniques of software fault tolerance. The proposed FSPN based modeling framework is novel in many aspects. First, the FSPN formalism itself, as proposed in [24], is extended by adding *flush-out* arcs. Second, the three techniques are simultaneously captured in a single model for the first time. Third, the formalism enables modeling dependencies of the three techniques on various system features such as failure, load and time in the same framework. Further, our base FSPN model can be viewed as a generalization of most previous models in the literature. To demonstrate, we present a set of FSPNs which are simple modifications of the base model. These represent software systems with checkpointing only, rejuvenation only and checkpointing and rejuvenation. We show that these FSPNs can not only mimic previously published models but can also extend them. For one FSPN model, we present numerical results to illustrate their usage in deriving measures of interest.

Keywords: Fluid Stochastic Petri Nets, System Aging, Rejuvenation, Checkpointing, Rollback Recovery.

1 Introduction

It is now well established that outages in computer systems are caused more due to software faults than due to hardware faults [22, 33]. Therefore, to build reliable systems, it is imperative to improve the reliability of software during the design, code development as well as the execution phase. Increasing the testing time proportionately increases development costs but provides only marginal gains. Moreover, it is well known that regardless of testing effort, large software always contains some residual bugs. Therefore, improving the execution reliability of software via cost-effective fault-tolerance techniques is becoming an attractive alternative.

One such technique called *software rejuvenation*, first proposed by Huang *et al.* in [25], is devised to tolerate a specific subset of software faults. These faults result in a steady accrual of error conditions in the internal state and/or the external operating environment of the executing software. The phenomenon is called *software aging* [25]. Effects of aging manifest as failures which may be observed as just performance degradation (for instance reduction in the service rate of a database server), fail-stop behavior (such as an application hang or a crash), or abnormal termination (such as erroneous output of a simulation). Memory leaks, unreleased object references, faulty pointer handling and roundoff errors are some typical examples of software faults which result in aging during software execution. Numerous real-life examples, evincing the widespread existence of aging in software systems ranging from popular desktop operating systems and applications to life and mission critical systems can be found in the literature. Interested reader is referred to [17] for a comprehensive list. Garg *et al.* [18] have shown the evidence of aging in general purpose UNIX systems via statistical time series analysis of resource usage data. Software rejuvenation was originally defined as "preemptive rollback of a running process to a clean state" and simply involved restarting a process after some cleanup. Several examples of the use of rejuvenation in real systems may be found in [21, 23, 25, 30, 34].

Checkpointing, which involves saving the execution state of a program, along with transaction logging, is another well known fault tolerance technique used primarily to reduce the recovery time after failures. In this sense, it is complementary to the failure masking property of redundancy techniques and the failure avoidance property of software rejuvenation. Naturally, combining checkpointing with rejuvenation, as proposed in [10, 13] yields greater benefits.

As mentioned earlier, the concept of rejuvenation was proposed originally to simply mean process restart which results in down time. However, at times, the system may undergo a procedure which does not involve down time yet changes the degraded state to a more cleaner one. This self-restoration only causes a performance overhead but no downtime. Well known examples include online garbage collection such as in *emacs* or *Java Virtual Machine (JVM)*, transaction logging and data backup and archiving.

A typical example where rejuvenation, checkpointing and restoration are used together is a database server. The database state is checkpointed to limit the size of "redo" logs and to reduce the recovery time after a failure. The database server process is occasionally restarted (rejuvenated), possibly on another machine to counteract aging. This planned outage not only prevents unplanned crashes, which might occur during periods of high load but also restores the service rate. Moreover, administrators occasionally cleanup the disk to free more space for the logs. The latter self restoration does not interrupt the service, yet counteracts aging by providing extra resources.

Rejuvenation implies that the system experiences downtime equal to the time it takes to cleanup and restart the software. However, once performed, the probability of an unplanned failure of the software is reduced. Taking checkpoints also implies that the software is unavailable for the duration it takes to complete a checkpoint. On the other hand, it saves the work executed so far thereby eliminating the need to reexecute it after a failure. Similarly, self-restoration, although it does not result in down time, the software experiences performance degradation for the duration. Therefore, an important issue in the use of these techniques is to evaluate the tradeoff of their benefit against the overhead they cause and determine when and how often checkpointing, rejuvenation or restoration should be initiated. Analytical modeling has been used to address this issue. We now briefly describe previous research work in modeling.

1.1 Previous Work in Modeling Checkpointing and Rejuvenation

In [15], Markov regenerative stochastic Petri net (MRSPN) formalism is used to model a software system with rejuvenation. Aging is modeled simply by a two-stage Hypo-exponential failure and no performance effects of aging are taken into account. The underlying Markov Regenerative Process with a periodic rejuvenation policy is solved to compute the availability and to determine the optimal "period" for initiating rejuvenation. Garg *et al.* [17] use a Non-homogeneous Continuous Time Markov Chain (NHCTMC) to model the behavior of a transactions based software system. The model allows generally distributed time to failure as well as its dependence on load and time, both current as well as cumulative. It also allows the service rate to be dependent on time and load factors. The NHCTMC is solved numerically to yield availability, throughput, probability of loss and an upper bound on the mean response time of transactions. Additional models towards evaluating the effectiveness of software rejuvenation can be found in [16, 31, 34]. A different system model is discussed in [5] which assumes that the degradation level of the system can be observed at predefined observation instances and rejuvenation is initiated based on the observed degradation level, rather than at periodic intervals.

In the past two decades there have been a number of papers which evaluate the fundamental tradeoff of reduction in recovery time and the checkpoint overhead itself and determine the optimal checkpoint interval in different software systems. In [7, 11, 27, 28, 32], systems with a finite failure free completion time are modeled in which checkpointing is used to either minimize the expected completion time or maximize the probability that the software completes execution within a certain deadline. In [6, 19, 20, 29], long running server software systems which employ checkpointing are modeled. The measures of interest in this case include availability, throughput and response time.

Expected completion time of a program with a finite mission time is computed in [13]. The stochastic model allows generally distributed time to failure and combines checkpointing, with rejuvenation. Performance degradation is not captured and under periodic rejuvenation and checkpointing policies, corresponding optimal intervals are determined.

In this paper, we apply the Fluid Stochastic Petri Net (FSPN) modeling formalism to analyze degrading (aging) software systems which employ rejuvenation and checkpointing. In the literature, there are several variations of Fluid Stochastic Petri Nets (FSPNs). In some cases this formalism has been used for deriving analytical solutions [24, 35, 36], others use simulation as a solution methods for FSPNs [2, 3, 9]. The FSPN formalism proposed in [24], extends the original proposal of [35], by allowing the fluid level in continuous places to affect the firing rate of the timed transitions and the rates of fluid flow into and out of continuous places. In that paper the rates of the timed transitions and the fluid flows are "deterministic" functions of the FSPN state (discrete and continuous). This formalism is referred to as "first order FSPNs". In [36], the continuous places are filled and drained at random (normally distributed) rate and this formalism is referred to as "second order FSPNs". In this paper we extend the formalism of "first order" FSPNs by introducing *flush-out* arcs [1]. These arcs connect continuous places to transitions, and describe the capability of a transition to flush out all existing fluid from a continuous place when it fires.

The FSPN framework proposed here coupled with the use of appropriate measures enables us to capture the dynamic behavior of the software. Our contributions in this paper include the following.

- The model presented in this paper captures rejuvenation, aging/degradation, checkpointing and self-restoration along with interdependencies in a combined general FSPN model.
- To capture these dynamics, we also enhance the FSPN modeling framework, as proposed by Horton *et al.* in [24] to include flushing of fluid places. That formalism allows for only draining the fluid out of a place at finite rate and discrete jumps in fluid levels are not possible to be modeled.
- FSPN model for a server system which employs both rejuvenation and checkpointing.
- The FSPN modeling framework which allows to capture realistic scenarios such as failure during any special operational phase (checkpointing, self restoration), and a combined system of checkpointing and rejuvenation, where checkpointing does not necessarily result in a system renewal. Most of the previous modeling work is limited in one or more of these aspects.
- Numerical analysis techniques for FSPNs with flush-out arcs and the definition of some performance measures have been included in the paper. This demonstrates that the proposed FSPN formalism is useful towards evaluating performance measures for systems with aging, rejuvenation, and checkpointing.

The rest of the paper is organized as follows. In Section 2, we introduce the FSPN modeling formalism and describe how it extends the existing ones. Section 3 consists of the description of the overall model. We also present the combined FSPN model which captures the dynamics of degradation, rejuvenation, self-restoration and checkpointing in this section. Our intent is to show that the extended FSPN formalism is capable of capturing the multiple fault tolerance mechanisms in a single model and therefore previous papers which attempt to model only a subset of these mechanisms are simply special cases of this general model. In Section 4, we present such special cases and relate them to specific previous modeling research. We also present the FSPN model of a server software which employs checkpointing and rejuvenation and constitutes a contribution. Section 5 consists of numerical experiments to illustrate the use of FSPN models and finally we close in Section 6 with conclusions and pointers to future work.

[1] Adding flush-out arcs results in non-conformance to the real physical property of fluids; that they can be pumped and drained only at a finite rate making fluid level a continuous function in time. Nevertheless we still use the name and the concepts of FSPNs to be consistent with the previous literature.

2 Fluid Stochastic Nets[1]

In this paper, we apply the FSPN formalism to model systems which employ rejuvenation and checkpointing and then we present the FSPN formalism necessary to this aim, that represents an extension of the one proposed by Horton *et al.* in [24].

We define FSPN as a 7-tuple $\langle \mathcal{P}, \mathcal{T}, M_0, \mathcal{A}, F, W, R\rangle$, where $\mathcal{P}$ is the set of places partitioned into set of discrete places $\mathcal{P}_d = \{p_1, \ldots, p_{|\mathcal{P}_d|}\}$, and the set of continuous places $\mathcal{P}_c = \{c_1, \ldots, c_{|\mathcal{P}_c|}\}$. The discrete places may contain tokens (the number of tokens in a discrete place is a natural number), while the marking of continuous places is a non negative real number. In the graphical representation, a discrete place is drawn as a single circle while a continuous place is drawn with two concentric circles. The set of transitions $\mathcal{T}$ is partitioned into the set of timed transitions $\mathcal{T}_e$ and the set of immediate transitions $\mathcal{T}_i$. We denote the timed transitions with uppercase letters and the immediate transitions with lowercase letters.
The state (marking) of an FSPN is described by a vector $M = (\boldsymbol{m}, \boldsymbol{x})$, where the vector $\boldsymbol{m}$ is the marking of the discrete places and and the vector $\boldsymbol{x}$ represents the fluid levels in the continuous places. We use $\mathcal{S}$ to denote the set of all possible states $(\boldsymbol{m}, \boldsymbol{x})$. The initial state of the FSPN is denoted by M_0.

The set of arcs $\mathcal{A}$ is partitioned into four subsets $\mathcal{A}_d$, $\mathcal{A}_c$, $\mathcal{A}_h$, and $\mathcal{A}_f$. The subset $\mathcal{A}_d$ contains the discrete arcs which can be seen as the function $\mathcal{A}_d : ((\mathcal{P}_d \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}_d)) \rightarrow \mathbb{N}$. The arcs $\mathcal{A}_d$ are drawn as single arrows. The subset $\mathcal{A}_h$ contains the inhibitor arcs, $\mathcal{A}_h : (\mathcal{P}_d \times \mathcal{T}) \rightarrow \mathbb{N}$. These arcs are drawn with a small circle at the end. $\mathcal{A}_c$ is a subset of $(\mathcal{P}_c \times \mathcal{T}_e) \cup (\mathcal{T}_e \times \mathcal{P}_c)$ and represents the set of continuous arcs. In the graphical form, arcs in $\mathcal{A}_c$ are drawn as double arrows to suggest a pipe. The subset $\mathcal{A}_f : (\mathcal{P}_c \times \mathcal{T}_e)$ contains the *flush-out* arcs. These arcs connect continuous places to transitions, and describe the capability of a transition to flush out all existing fluid from a continuous place when it fires. This is an extension to the FSPN formalism proposed in [24] in which such discrete jumps in the fluid level are not allowed. The dynamics of a real system, such as loss of queued jobs in a server or work done, require this extension for modeling purposes. In other words, the previous FSPN formalism is insufficient to model such behavior. In the graphical representation, arcs in $\mathcal{A}_f$ are drawn as thick single arrows.
The rules that define when a transition is enabled, are the same as the usual ones, defined in [1], i.e., the enabling conditions depend only on the discrete part of the FSPN. From this it follows that fluid and flush-out arcs does not change the enabling conditions of a transition because they act only the continuous part of the marking.
In our formalism, an enabled transition $T_j \in \mathcal{T}_e$ may drain fluid out of its continuous input places, and may pump fluid into its continuous output places at a finite rate. The rates of flow may be dependent on the discrete as well as the continuous part of the state $M = (\boldsymbol{m}, \boldsymbol{x})$.

The firing rate function F is defined for timed transitions $\mathcal{T}_e$ so that $F : \mathcal{T}_e \times \mathcal{S} \rightarrow \mathbb{R}^+$. Therefore a timed transition T_j enabled in a marking $\boldsymbol{m}$ (discrete part of $M = (\boldsymbol{m}, \boldsymbol{x})$), fires at rate $F_j(M)$, allowing the firing rates to be dependent on the discrete and/or the continuous marking.
The weight function W is defined for immediate transitions $\mathcal{T}_i$ such that $W : \mathcal{T}_i \rightarrow \mathbb{R}^+$. Thus if an immediate transition t_i is enabled in a (vanishing) marking $\boldsymbol{m}$ (restricted only to the discrete marking because the immediate transitions do not dependent on the continuous part of the marking), it fires with probability $W(t_i)/\sum_{t_j \text{enabled in } \boldsymbol{m}} W(t_j)$.
In order to describe the evolution of the continuous part of the marking, we need to introduce the semantic of the continuous arcs $\mathcal{A}_c$. Each continuous arcs that connects a fluid place $c_l \in \mathcal{P}_c$ to an enabled timed transition $T_j \in \mathcal{T}_j$, causes a "change" in the fluid level of place c_l. The absolute rate of this change is determined by the value of a function R, called *flow rate function*. This is a function of both the arc and the state such that $R : \mathcal{A}_c \times \mathcal{S} \rightarrow \mathbb{R}^+ \cup \{0\}$. We assume that R is continuous on the fluid component of the marking. The sign of R is positive if the arc is directed toward the fluid place (i.e. (T_j, c_l)), and negative if it is directed toward the transition (i.e. (c_l, T_j)). Let $\mathcal{M}(\tau)$ be the marking process and $X_l(\tau)$ be the fluid level of place c_l at time τ. If no flush out occurs, and no

[1] The formalism used in this paper is applied for different purposes in an ongoing work of some of the authors.

boundary is reached (i.e., there are no empty fluid places), then:

$$\frac{dX_l(\tau)}{d\tau} = r_l(\mathcal{M}(\tau)) = \sum_{T_j \text{enabled in } \mathcal{M}(\tau)} R_{j,l}(\mathcal{M}(\tau)) - \sum_{T_j \text{enabled in } \mathcal{M}(\tau)} R_{l,j}(\mathcal{M}(\tau)). \tag{1}$$

The function $r_l(\mathcal{M}(\tau))$ in Equation (1) represents the total rate of fluid change in state $\mathcal{M}(\tau)$. If a transition T_j fires at time τ, and it is connected to a fluid place c_l with a flush out arc (i.e., $(c_l, T_j) \in \mathcal{A}_f$), or a boundary is reached when $r_l(\mathcal{M}(\tau))$ is negative (i.e., $X_l(\tau) = 0$ and $r_l(\mathcal{M}(\tau)) < 0$), then:

$$X_l(\tau) = 0.$$

It must be pointed out that in some previous fluid formalism (see for instance [12]) $r_l(\mathcal{M}(\tau))$, can be a piecewise continuous on the fluid component of the marking. In this paper we provide equations and solutions technique only when R is continuous on the fluid component of the marking.

2.1 Performance measures defined on a FPSN

Previous work on FSPNs was mainly concentrated on the analytical or simulative description of the dynamic of the system. Little attention has been paid to the modeling power of the formalism and to the investigation of the meaning of the performance measures that can be obtained from the analysis of the model. Here, we attempt to classify the set of performance measures which can be associated with FSPN models. The limit of the modeling abilities of FSPNs is still an open research area.

It can be said, in general, that the set of performance measures that can be evaluated from a FSPN encompasses the set of those that can be evaluated in discrete SPN models. In fact, in addition, we can define new measures that are specifically related to the fluid (or continuous) part of the net. We can refer to the measures connected to the discrete part of the FSPN as *discrete performance measures* and to those connected to the continuous part as *continuous performance measures*.

Moreover, *discrete performance measures* can still be classified as *discrete state measures* (when the measure refers to the probability of occurrence of some condition on the discrete markings) and *throughput measures* (when the measure refers to the passage of tokens through the net or to the number of firings of a transition). Similarly, *continuous performance measures* can be classified as *fluid state measures* and *flow measures*. Flow measures can be considered as the continuous counterpart of discrete throughput measures. The rate of flow through a fluid arc is the counterpart of the throughput of a discrete arc connected to a timed transition while the rate of flow through a flush-out arc is the counterpart of the throughput of an arc connected to an immediate transition. Since in FSPNs, the firing rate of a timed transition may depend both on the discrete and the continuous component of the marking, the firing time may be any generally distributed random variable, and the meaning and the evaluation of the throughput measures are more complex than in the Markovian SPN models.

A very elegant and unifying way to define and to compute both kinds of performance measures in discrete SPNs is by means of the concept of reward [4, 8]. In FSPN, the flow rate assigned to a continuous arc may, as well, be interpreted as a reward rate that can be dependent on the discrete and the continuous component of the marking. In this view, fluid places are structural elements whose fluid level represents the accumulation of the reward as a function of the time. Hence, in FSPN the reward is directly associated with the graphical representation of the model allowing to describe and evaluate all the reward measures in a natural way at the level of the graphical representation without using any additional specification (as in discrete SPN).

Furthermore, reward measures can be defined at a given time instant, in steady state (if it exists) or over a time interval. In Markov Reward Models, like those generated from Discrete SPNs, the reward measures that can be evaluated at the same cost of the solution of the standard Markov equation, are the expected instantaneous reward measures (either at a given time instant or in steady state) or the expected accumulated reward measures [4]. The majority of the packages dealing with SPNs restricts the evaluation of significant measures to the expected reward measures mentioned above.

The evaluation of the *cdf* of the reward accumulated over a finite time interval requires a considerable increase in the computational effort and is usually not offered in SPN packages. On the contrary, FSPNs allow to define these distribution measures within the default structural specifications and to evaluate them with the default modeling abilities. As an example, the *cdf* of the reward accumulated over a time interval can be evaluated as the distribution of the fluid level at a properly defined fluid place. Moreover, in order to evaluate the distribution of the completion time or response time of a task on a server, a suitable absorbing condition must be structurally defined on the FSPN, so that the above distribution can be computed as the probability of absorption in a structural element of the FSPN (an example of such a construction is given in Section 4.3).

3 System description

We consider a software system which exhibits aging/degradation in two ways.
Soft failure: The system performance decreases due to system degradation. An example is increased paging activity by the kernel due to locked memory resources which results in a reduction in effective CPU cycles. The user perceived effect may include less total work done per unit time, reduction in the service rate of a server etc.
Hard failure: The probability of the occurrence of a crash failure (failure rate) increases with time due to system degradation. The software becomes unavailable resulting in no work being done. Aging may result in soft, hard or both kinds of failures although one may be more noticeable than the other.
To counteract the effect of aging, software rejuvenation is used, whereas, to prevent the loss of work, checkpointing with rollback recovery is employed. In addition, the system may have complementary restoration capability which reduces the "age" (degradation level) without making the software unavailable.

We present an FSPN model of such a software system in which all the three are coexistent, which, to the best of our knowledge, have not been coexistent together before. We now list the processes, which together capture the dynamics of the software system and allow us via the FSPN formalism to evaluate various measures of interest. The FSPN model, shown in Figure 1, will be explained in the sequel.

- *Degradation.* The degradation process, which models aging, is modeled by a continuous quantity which may depend on the number of jobs currently in the server queue. We assume that the software system is a server which serves customers arriving according to a Poisson process. The degradation may also simply depend on the time the software witnessed a renewal event (crash or rejuvenation) or it may depend on the total amount of work completed since the last renewal event. Our model can capture each of these dependencies.

- *Rejuvenation.* We assume that the decision of performing a rejuvenation may depend on the degradation level and on the time spent since last renewal event. It is natural to assume that a rejuvenation always forces a checkpoint otherwise work already done since the last checkpoint is lost.

- *Work.* A continuous quantity, simply captures the work done by the system. An example is the total CPU cycles used by the server. This work is occasionally saved with a checkpoint. If a crash occurs the work done by the system not saved yet is lost.

- *Time*, also a continuous quantity is needed to keep track of the time spent since the last checkpoint, crash or rejuvenation occurred and is needed to model dependencies as well as to calculate measures of interest.

- *Checkpoint.* When a checkpoint occurs the work done by the system not saved yet is saved. A crash can occur during a checkpoint, in this case the work not saved by a previous checkpoint is lost.

- *Crash.* When the system crashes the work done by the system not saved yet by a checkpoint is lost. A crash is a renewal event, i.e., it resets the degradation level of the system.
- *Self Restoration.* We assume that the self restoration capability of the software, when in progress, continually decreases the degradation level.
- *Workload.* This is used to represent the service behavior of the system. The service time may depend on the degradation level and on the number of customers in the system. We assume that the number of customers that can be accepted by the system is limited by a finite buffer size. When the buffer is full, during a crash or a rejuvenation the arrival process is stopped. On the other hand the service stops during checkpoints, rejuvenations, and crashes.

The repair after a crash failure and rejuvenation is assumed to renew the system, i.e., the system is restored to the "as good as new" state. Crash failure may induce a loss of customers in the system and the performed work since the last checkpoint. We now proceed to describe the FSPN model, shown in Figure 1, which captures the above aspects together. In Section 4, we will present special cases of this model which capture a subset of these aspects in more details.

3.1 An FSPN for model of the considered system

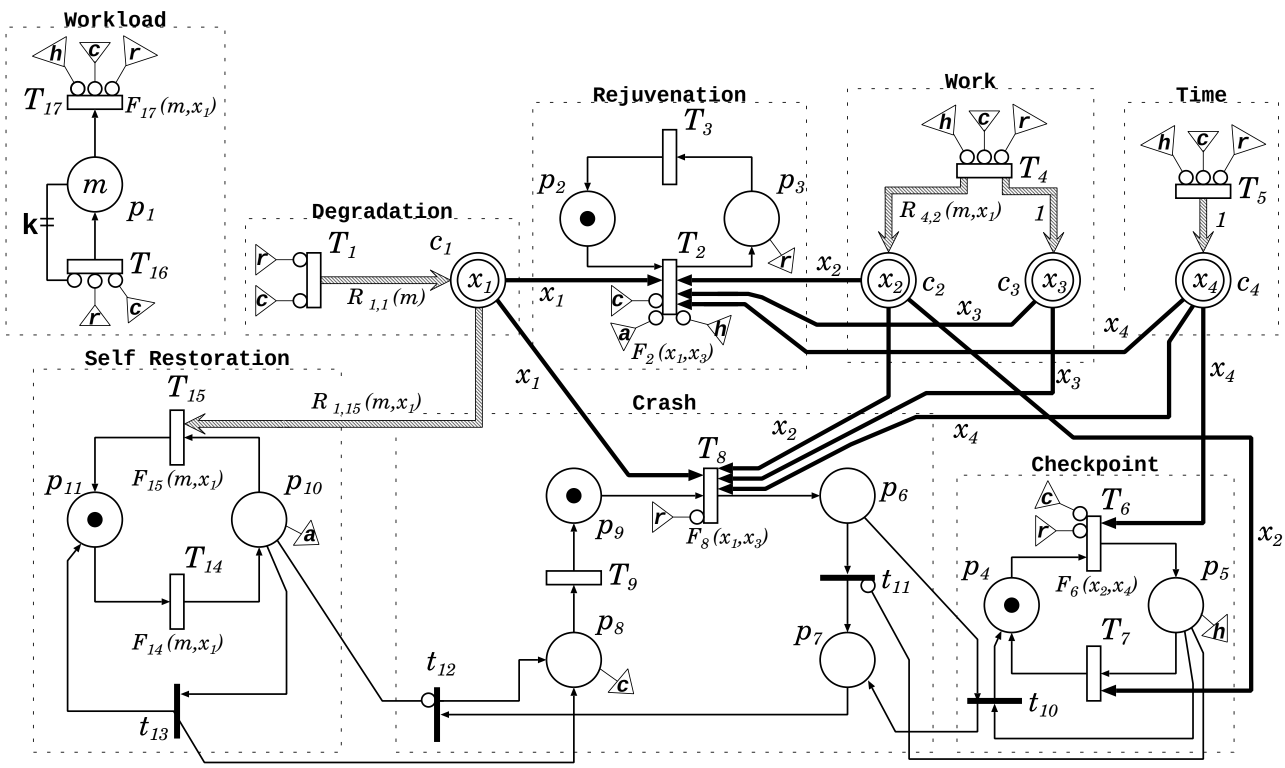

Figure 1: FSPN model of a server system with rejuvenation and checkpoints and self-restoration

Table 1 maps the labels on the transitions and on the places of the FSPN of Figure 1 to the processes described earlier. In order to make the net representation clearer, we have denoted some inhibitor arcs with a triangle and a label which implies that there are inhibitor arcs from place to transitions having the same label.

- We start by describing the subnet labeled *Degradation*. The continuous marking of place c_1, i.e., x_1 is a measure of the degradation level of the system. Transition T_1 pumps fluid in place c_1 and represents the increasing of the system degradation. The flow rate at which the fluid is pumped in place c_1 can depend on the (discrete) marking of place p_1 that represents the number of customers in the system (e.g, $R_{1,1}(m)$). If we need to represent a degradation process that

Transictions	Activities
T_1	Increase of degradation level
T_2	Beginning of a rejuvenation (preceded by a forced checkpoint)
T_3	End of a rejuvenation
T_4	Increase of work and the time spent since last renewal event
T_5	Increasing of the time spent since last checkpoint
T_6	Beginning of a checkpoint (independent of a rejuvenation)
T_7	End of a checkpoint
T_8	Occurrence of a crash
T_9	Recovery from a crash
t_{10}	Crash during a checkpoint
t_{11}	Crash independent of a checkpoint
t_{12}	Crash independent of self restoration
t_{13}	Crash during a self restoration
T_{14}	Beginning of a self restoration
T_{15}	End of a self restoration
T_{16}	Arrival of customers to the system
T_{17}	Service of customers
Fluid Place	**Meaning**
c_1	Degradation level (continuous marking denoted by x_1)
c_2	Work not saved yet (continuous marking denoted by x_2)
c_3	Time spent since last renewal event (continuous marking denoted by x_3)
c_4	Time spent since last checkpoint (continuous marking denoted by x_4)

Table 1: Description of the FSPN of Figure 1

depends on the time since last renewal event (rejuvenation or crash) we define the flow rate at which the fluid is pumped in place c_1 as $R_{1,1}(m, x_3)$ (where x_3 is the continuous marking of place c_3 and represents the time since last renewal event). We can also express a non-linear degradation processes by making the flow rate at which the fluid is pumped in place c_1 function of x_1 itself (e.g., $R_{1,1}(x_1)$). dependent on x_1 itself.

- The subnet labeled *Rejuvenation* represents the rejuvenation events. In the initial marking, place p_2 contains 1 token and p_3 no tokens. Transition T_2 represents the beginning of a rejuvenation, and its firing rate $F_2(x_1, x_3)$ may depend on the degradation level of the system (x_1) and on the time since last renewal event (x_3). We assume that when the system performs a rejuvenation, a checkpoint is forced. The vice versa is not true, i.e., a checkpoint does not imply a rejuvenation. When T_2 fires (a rejuvenation begins) the token in place p_2 is moved in place p_3 and places c_1, c_2, c_3, and c_4 are flushed out (thick arcs from these places to transition T_2). In this manner the firing of T_2 resets the level of degradation of the system, the work not saved yet, the time since last renewal event, and the time since last checkpoint to zero. Transition T_3 represents the time required to complete a rejuvenation. When the system is performing a rejuvenation (token in place p_3) the following transitions are disabled: T_1, T_4, T_5, T_6, T_8 and T_{16}. This is obtained with inhibitor arcs with label r. The meaning is that while the system is rejuvenating all the jobs and timers are stopped and the system is as good as new at the end of the rejuvenation. Moreover neither a checkpoint (since rejuvenation is already a checkpoint) nor a crash (since its occurrence can be included in transition T_3 that models the time spent by the system in this state) can occur.

- The subnet labeled *Work* describes the accumulation of the work done by the system. Transition T_4 pumps fluid in places c_2 and c_3 with rate $R_{4,2}(m, x_1)$ and 1 respectively. The fluid level of c_2 represents the work of the system not saved yet by a checkpoint, while the fluid level of c_3 represents the time since last renewal event (i.e., crash or rejuvenation). The rate $R_{4,2}(m, x_1)$ at which T_4 pumps fluid in c_2 may be dependent on the degradation level of the system (continuous marking x_1) and on the number of customers in the system (discrete marking m). In this way, we can express "soft failures" and system load dependency. Place c_2 is flushed out by the firing of transitions T_2 (checkpoint forced by a rejuvenation), T_7 (execution of a checkpoint without rejuvenation), and T_8 (occurrence of a crash). Transition T_4 is disabled when the system is performing a rejuvenation or a checkpoint, or when a crash occurs. This is obtained with the inhibitor arcs labeled c, h, and r. Place c_3 is flushed out by firing of transitions that represents the occurrence of a renewal event, i.e., rejuvenation or crash.

- The subnet labeled *Time* describes the time spent since the last checkpoint. The continuous marking of place c_4 (denoted by x_4) represents this quantity. Place c_4 is flushed out by the firing of a transition that represents the occurrence of one of the following events: rejuvenation (and checkpoint forced), checkpoint, and crash. The inhibitor arcs on transition T_5 represent that the time is stopped when either a checkpoint, rejuvenation or crash recovery is in progress.

- The subnet labeled *Checkpoint* describes the occurrence of a checkpoint independent of a rejuvenation. Transition T_6 represents the beginning of a checkpoint, the firing time $F_6(x_2, x_4)$ of this transition may depend on the quantities x_2 and x_4 that are the markings of place c_2 and c_4. This means that the occurrence of a checkpoint may depend on the quantity of work executed and not saved by a checkpoint yet, and by the time since the last checkpoint. The firing of T_6 flushes out place c_4 while T_7 flushes out place c_2. With these actions we represent the fact that the beginning of a checkpoint resets the time spent since the last checkpoint, while the completion of the checkpoint saves the work not saved yet. In this manner we can model the occurrence of a crash during a checkpoint. If a crash occurs (token in place p_6) after the beginning of a checkpoint (token in place p_5), transition T_7 cannot fire because it is enabled in conflict with the immediate transition t_{10}. The firing of t_{10} puts the token in place p_4. Place c_2 is not flushed out by transition T_7 that represents the saving of the work not saved yet, but it is flushed out by transition T_8 that represents the lost of this work. The inhibitor arcs on transition T_6 inhibit the occurrence of a checkpoint independent of a rejuvenation during a rejuvenation or when a crash occurs. When the system is performing a checkpoint (token in place p_5) the occurrence of a rejuvenation is inhibited (label h on place p_5 and on transition T_2).

- The subnet labeled *Crash* describes the crash event. Transition T_8 represents the occurrence of the crash. The firing time $F_8(x_1, x_2)$ of this transition may depend on the degradation level (x_1) and on the time spent since last renewal event (x_3). The immediate transitions t_{10} and t_{11}, enabled in mutual exclusion, model the situation when a crash occurs during a checkpoint phase (t_{10}). The immediate transitions t_{12} and t_{13}, enabled in mutual exclusion, model the situation when a crash occurs during a self restoration phase (t_{13}). The occurrence of a crash resets the degradation level (place c_1), the level of work not saved yet (place c_2), the time spent since last renewal event (place c_3), and the time spent since last checkpoint (place c_4). The inhibitor arc from place p_3 to transition T_8 inhibits the occurrence of a crash during a rejuvenation.

- The subnet labeled *Self Restoration* describes some self adjusting capability of the system. This can be seen as a light rejuvenation, i.e., a partial restoration that does not prevent the normal behavior of the system. With this subsystem we can model for example a garbage collector. Transition T_{14} represents the beginning of this partial restoration. The firing rate of T_{14} $F_{14}(m, x_1)$ may depend on the number of customers within the system and on the degradation level. During the self restoration phase (token in p_{10}) a crash can occur (firing of T_8 followed by the firing of t_{11}), but the occurrence of a rejuvenation is inhibited (inhibitor arc from place p_{10} to transition T_2). The duration of the self restoration phase, modeled by transition T_{15}, may depend on the level of degradation and on the number of customers in the system.

- The subnet labeled *Workload* models the arrival of customers in the system (firing of transition T_{16}) that require service (firing of transition T_{17}). The service time may be dependent on the degradation level and on the number of customers in the system. The arrival of customers is inhibited when the buffer is full, i.e., there are k customers in the system (inhibitor arc with multiplicity k from place p_1 to transition T_{16}), when a crash or a rejuvenation occur (inhibitor arcs with labels c and r on transition T_{16}). On the other hand the service stops during a checkpoint, a crash, and a rejuvenation (inhibitor arcs with labels h, c, and r on transition T_{17}).

3.2 Performance measures

In this section, we propose some interesting performance measures for the FSPN of Figure 1, according to the taxonomy proposed in Section 2.1, and we discuss how to evaluate them.

Examples of *discrete state measures* are, for instance, the point or the steady state availability (probability that the system is working at a given time instant or in steady-state) or the interval availability (the expected fraction of time in the interval $[0, \tau)$ in which the system is working). From the FSPN of Figure 1, these measures can be obtained by summing up the probability of the markings whose discrete component carries a token in places p_2, p_4, and p_9. Further, let us denote by P_{loss} the long run probability that an arriving customer will be lost. P_{loss} can be computed by observing that the considered event can happen for two different reasons: *i)*- the system is not available due to a rejuvenation (no tokens in place p_2) or a crash (no tokens in place p_9); *ii)* - the buffer is full (k tokens in place p_1).

The expected response time of a customer, T_{res} can be defined as the expected time that a customer spends in the system, and can be evaluated from a combination of a *discrete state* measure and a *throughput* measure. Indeed, applying the Little's law, T_{res} can be obtained by computing the average number of tokens in place p_1 and the throughput of transition T_{17}. If the firing rate of T_{17} is marking independent, or it depends on the discrete part of the marking only, this measure can be computed by standard techniques. If, instead, the firing rate of T_{17} depends on the continuous component of the marking, the underlying marking process is no more a Markov chain, but computation of T_{res} can still be obtained in the FSPN formalism with a reasonable effort (computation of the probability of the complete markings).

Another measure which can be evaluated based on *discrete state* measures is the completion time, i.e., the time needed to complete a given amount of work. It will be shown in Section 4, that introducing a little modification to the FSPN of Figure 1, we can compute the expected value and the *cdf* of the completion time. The possibility of evaluating the *cdf* of the completion time enables us to evaluate other related performance measures such as the probability of completing a task by a given deadline which is a fundamental performance characterization for real-time systems. This last result represents an improvement with respect to the measures that can be computed using the approach described in [13].

The set of *continuous performance* indices that can be computed for systems with rejuvenation and checkpointing is based on measures that are related to the continuous component of the FSPN. In the proposed formalism, it is possible to compute the flow rates along fluid arcs, the flow rates along flush-out arcs and the service rates of timed transitions with firing rates dependent from both the discrete and the continuous component of the marking. An examples for these measures can be represented by the portion of "useful" work done with respect to the total work performed by the system. This measure can be obtained by computing the flow rate along flush-out arcs (as it will be shown extensively in Section 5).

4 FSPN models of special systems

In Section 3 a general system model is presented which captures the effect of checkpointing self restoration and rejuvenation at the same time. Some details specific to a particular system, however, is not captured in the model of Figure 1 to keep it at reasonable complexity. In this section we provide detailed models of degrading systems where only a subset of these fault tolerance schemes are applied. At the same time we indicate how the introduced models can capture the systems discussed in the previous literature.

4.1 System with checkpointing and rollback recovery only

A FSPN model of a system with checkpointing and rollback recovery is depicted in Figure 2. Note that the *crash* subnet of this model is more detailed than the *crash* subnet of original FSPN in Figure 1.

In the case of a crash failure (T_8 fires), a token reaches p_7 and system repair starts. When the repair completes (firing of transition T_{12}) the rollback recovery phase starts whose time depends on the amount of logged transactions since the last checkpoint (fluid level of c_2).
Common assumptions of some papers dealing with checkpointing are that the system fails at a constant

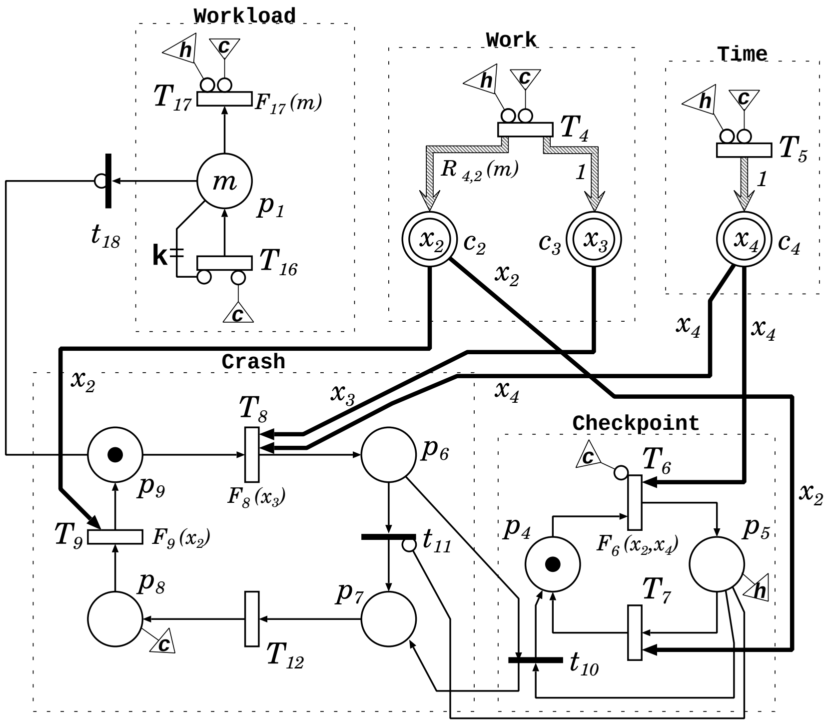

Figure 2: FSPN modeling a system with only checkpointing

rate (exponentially distributed failure time), and that the system renews after every checkpoint. This case can be captured by eliminating T_5 and c_4, and the associated arcs from Figure 2.

As long as the system failures were mainly caused by hardware failures the exponential failure time assumption was widely accepted. When the software problems become dominant in system failures the exponential failure time assumption was relaxed. Some models still assumed system renewal at checkpoints [11], while some others assumed that system degradation is accumulated till the occurrence of a crash failure [20]. Due to the independent "clocks" at c_3 and c_4, the FSPN model in Figure 2 can capture both assumptions because the time spent since the last checkpoint and the time spent since the last system failure are represented by the markings of c_4 and c_3 respectively.
General failure time distributions (including exponential) can be captured in our model by assigning appropriate rate function, $F_8(\cdot)$ to the transition T_8. The renewal of the failure process at checkpoint can be captured by making $F_8(\cdot)$ a function of the fluid level in c_4, which represents the time spent since the last checkpoint. Moreover, by making $F_8(\cdot)$ a function of fluid level in c_3 (which represents the time spent since the crash) the failure process may be assumed to renew only upon a crash. In other words, degradation continues through checkpointing. In some cases the real system behavior is better captured assuming that the failure rate is a function of the work done since the last checkpoint. For instance, no degradation might occur if the CPU is idle. This case can be easily captured by making the transition rate of T_8 dependent on the fluid level in c_2.

Another common assumption in previous models, for the sake of analytical tractability, is that failures do not occur during checkpointing. The FSPN model in Figure 2 allows failures to occur during checkpointing which is closer to the real behavior.

4.2 System with only rejuvenation

Rejuvenation is used primarily to reduce the effect of aging, which results in an increasing (crash) failure rate [25]. Different time to failure distributions (Hyper-exponential in [25]) and rejuvenation intervals (exponentially distributed in [25], deterministic in [15]) have been considered. The FSPN in Figure 3 captures these models by assigning appropriate transition rate functions $F_8(\cdot)$ and $F_2(\cdot)$ to the transition T_8 and T_2, respectively.

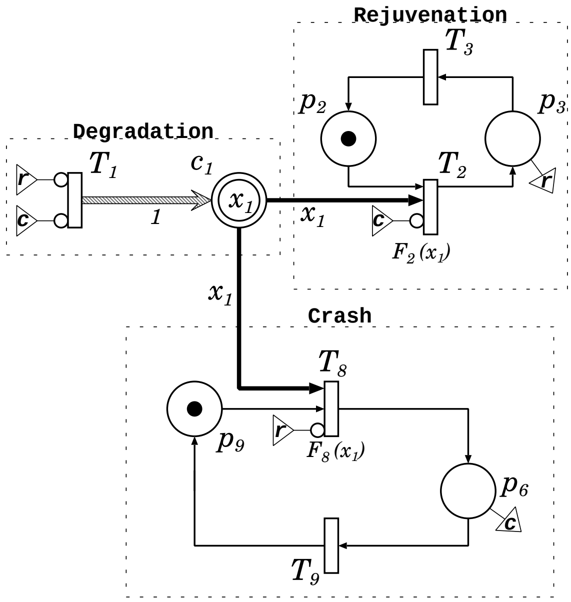

Figure 3: FSPN modeling a system with only rejuvenation

In [14, 16, 17] the system was assumed to be a long running server software prone to only crash failures and which serves randomly arriving customers. In these systems, rejuvenations and crash failures flush out the customers from the system. Moreover, any customer arriving during recovery or rejuvenation is also lost. Therefore, the number of lost customers becomes a performance measure of interest to determine and to optimize. Two policies were proposed to minimize the number of lost customers. Policies which adopted equidistant rejuvenation intervals independent of the number of customers in the system at the beginning of rejuvenation are referred to as time dependent policies. The FSPN in Figure 4 captures the time dependent rejuvenation policy when the service rate of transition T_2 is independent of m (marking of place p_1) but dependent on the fluid level of place c_1 ($F_2(x_1)$). In case of a crash failure or a rejuvenation the customers are flushed out from the system by the immediate transitions t_{18} and t_{19} respectively. The probability that a customer is lost may be reduced by initiating rejuvenation only when the system is empty in addition to a periodic interval. The rejuvenation policies which takes into account the number of customers in the system are referred to as time and load dependent policies. A simple time and load dependent policy was studied in [14] which uses two thresholds to control rejuvenation. Rejuvenation is initiated (T_2 fires) as soon as the first threshold expired and there is no customer in the system (p_1 is empty, and t_{21} fires), or when the second threshold expires. This behavior is captured by appropriate $F_2(m, x_1)$. In our model, soft failures, i.e., the degradation in system performance, can be additionally considered by making the service rate, transition T_{17}, depend on the degradation level of the system.

A system with only soft failures (performance degradation) and rejuvenation is studied in [31] and an optimal rejuvenation policy based on the number of customers in the system and known service rate function is proposed. This behavior can be obtained by eliminating the *Crash* subnet from Figure 4, where the dependence of the system performance on the degradation level is described by the transition rate $F_{17}(m, x_1)$ which depends on the fluid level in c_1.

Finally, the combined effect of increasing (crash) failure rate and decreasing system performance was considered in [17] and The FSPN in Figure 4 mimics this model where system aging is captured by $F_8(x_1)$ (hard failure) and $F_{17}(m, x_1)$ (soft failure).

4.3 System with Checkpointing and Rejuvenation

From the above discussion, it is clear that a periodic renewal of aging systems can increase their performance with respect to some performance measures. It is also straightforward that the state of continuously running servers has to be saved before rejuvenation, i.e., a checkpoint has to be made

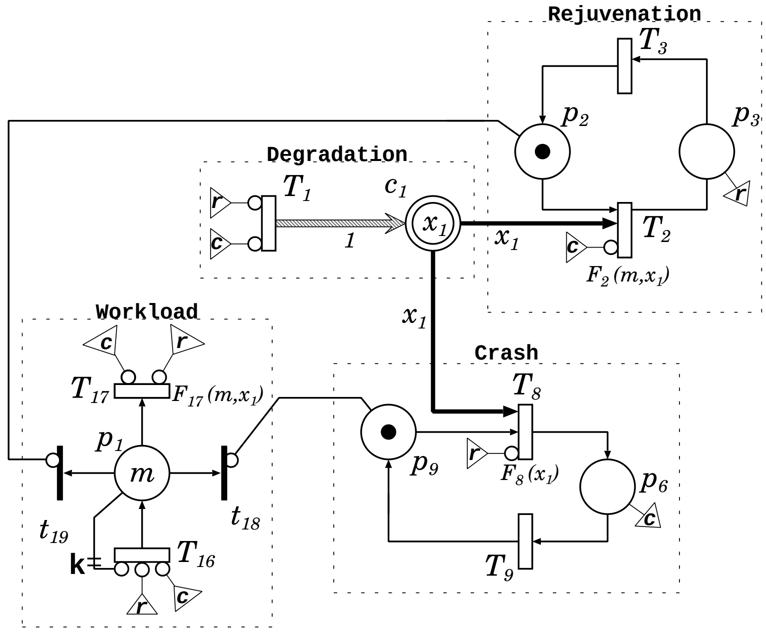

Figure 4: FSPN modeling a system with only rejuvenation and the external process

before rejuvenating the system. In [13], a software with a finite failure free completion time is modeled and it is shown that additional performance gain can be obtained, i.e., the expected completion time can be further reduced if the system state is saved more frequently than it is rejuvenated. The system can be modeled via our FSPN framework as well and is shown in Figure 5. In addition, our model allows for failures to occur while checkpointing is in progress. This aspect is captured by immediate transitions t_{10}, t_{11} and places p_6 and p_5 along with associated input, output and inhibitor arcs. In addition, the FSPN in Figure 5 may be used to compute not only the average completion time but also the distribution of the completion time. The latter enables us to evaluate other performance measures such as the probability of completion by a given deadline which is relevant to real time systems. The subnets labeled *Total work* and *Completion* (together) model the finite failure free work needed to be completed by the software. Firing of transition T_{14}, controlled by the marking, x_5 of place c_5 denotes completion. Note that the rollback recovery time, determined by the firing rate of transition T_9 is a function of the marking x_2 of place c_2, which represents the work lost due to this failure.

It is natural to think of using checkpointing along with rejuvenation to enhance the availability and increase the throughput of a long running server software as well. This combination has not been modeled in any of the previous work and is easily obtained from the general FSPN model of Figure 1 by eliminating the part of self restoration and adding specific details to the external process. Figure 6 depicts the FSPN model.

As before, crash failures can occur during checkpointing which results in the loss of work as indicated by the marking x_2. The time since last rejuvenation or checkpointing, indicated by the marking x_4 is used to model the time dependence of checkpointing or rejuvenation policy by making the transition rates of T_2 and T_6 dependent on x_4. In addition, rejuvenation is initiated either when the system is empty or when a fixed interval expires. This is obtained with the rate dependence of transition T_5, this transition may fire when a given degradation level is reached and the system is empty (dependency on m and x_1) or when a fixed interval expires (dependency on x_4). This behavior, similarly to the one explained for the FSPN of Figure 4, is captured by a proper setting of $F_2(m, x_1, x_4)$. All jobs are flushed from the queue if either T_2 fires (rejuvenation) or if the transition T_8 fires (crash). Moreover, we assume that the service process (transition T_{17}) stops during checkpointing. However, arrival is not inhibited and jobs continue to be queued. Crash or rejuvenation, on the other hand inhibit both arrival as well as service processes. Typical measures of interest are availability of the server (probability of markings with one token in places p_2, p_4, and p_9), throughput of the system (throughput transition of T_{17}), useful work performed by the system (this can be computed by with the ratio between the sum

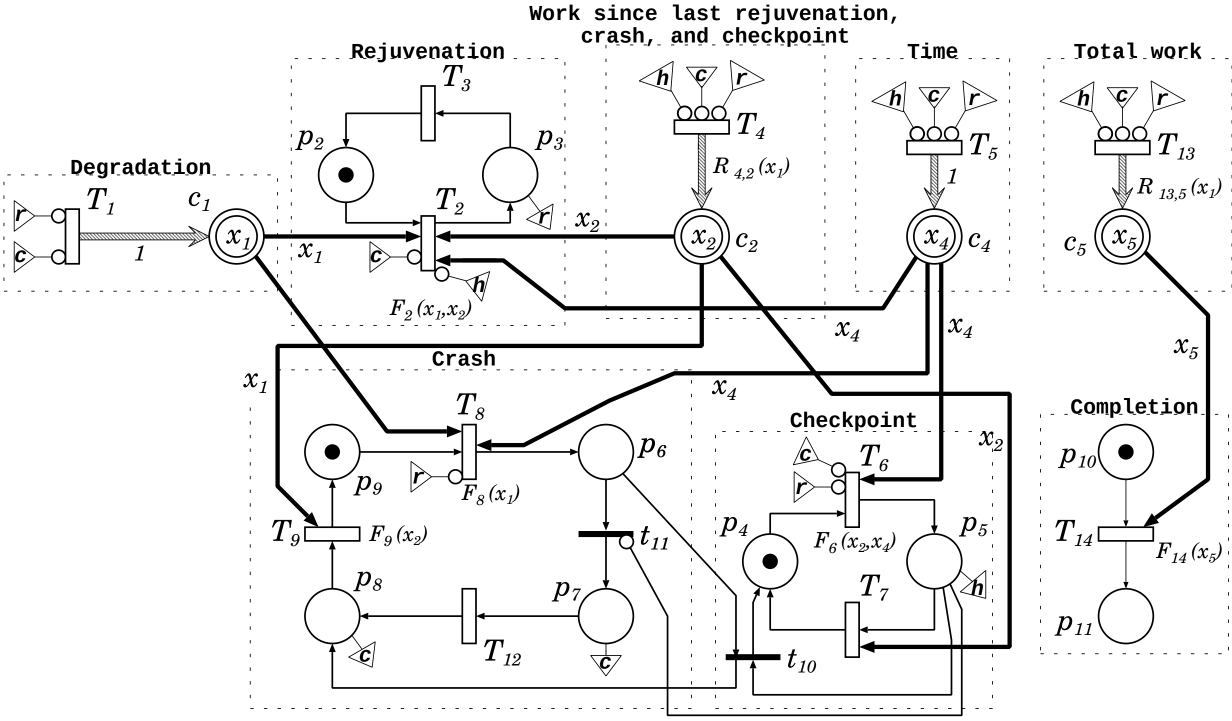

Figure 5: Modeling finite mission time system with rejuvenation and checkpointing

the flow rates of the flush-out arc connecting place c_2 to transition T_2 and transition T_7 and the total flow rate that exits form place c_2), etc.

5 Numerical Experiments

In this section, numerical results are derived for the net of Figure 7. We point out that these results, and also the technique to obtain them, are included to show how some performance measures can be obtained from a given FSPN. The net of Figure 7 models a system with checkpointing and rejuvenation. When the system performs a rejuvenation a checkpoint is forced, on the other hand a checkpoint does not imply a rejuvenation.

The rate of degradation is constant, $R_{11} = 1$, while the rate at which the work is accumulated depends on the degradation level ($R_{4,2}(x_1)$). The firing rates of the transitions associated with rejuvenation and crash failure (T_2 and T_8, respectively) also depend on the degradation level ($F_2(x_1)$, $F_8(x_1)$). The firing of transition T_6 represents the beginning of a checkpoint and depends on the accumulated work and the time elapsed since the last checkpoint ($F_6(x_2, x_4)$). The firing rates of timed transition T_3, T_7, and T_9 are constant and denoted by λ, γ, and μ, respectively.

In the following, we present the state equations describing the evolution of the marking process and some additional equations for the evaluation of specific performance measures. The analytical description of *flow measures* of FSPN models were not discussed previously in the literature, hence it can be considered as a contribution of this paper.

The four tangible (discrete) markings of the net are as follows: $\boldsymbol{m_0} = \{p_2, p_4, p_9\}$ denotes the working state (Normal) state of the system, $\boldsymbol{m_1} = \{p_3, p_4, p_9\}$ is the marking in which the system performs rejuvenation (Rejuvenation), $\boldsymbol{m_2} = \{p_2, p_5, p_9\}$ is the checkpointing state (Checkpoint), $\boldsymbol{m_3} = \{p_2, p_4, p_8\}$ denotes the system state reached when a crash failure occurs (Crash). To describe the probability of a given marking $(\boldsymbol{m}, \boldsymbol{x})$ we will derive the appropriate density functions. In $\boldsymbol{m_0}$ all the fluid levels may be nonzero:

$$\pi_0(\tau,x_1,x_2,x_4) = \lim_{\Delta_1,\Delta_2,\Delta_3 \to 0} \frac{\Pr(\boldsymbol{m}(\tau){=}\boldsymbol{m_0}, X_1(\tau) \in (x_1,x_1+\Delta_1), X_2(\tau) \in (x_2,x_2+\Delta_2), X_4(\tau) \in (x_4,x_4+\Delta_3))}{\Delta_1\ \Delta_2\ \Delta_3};$$

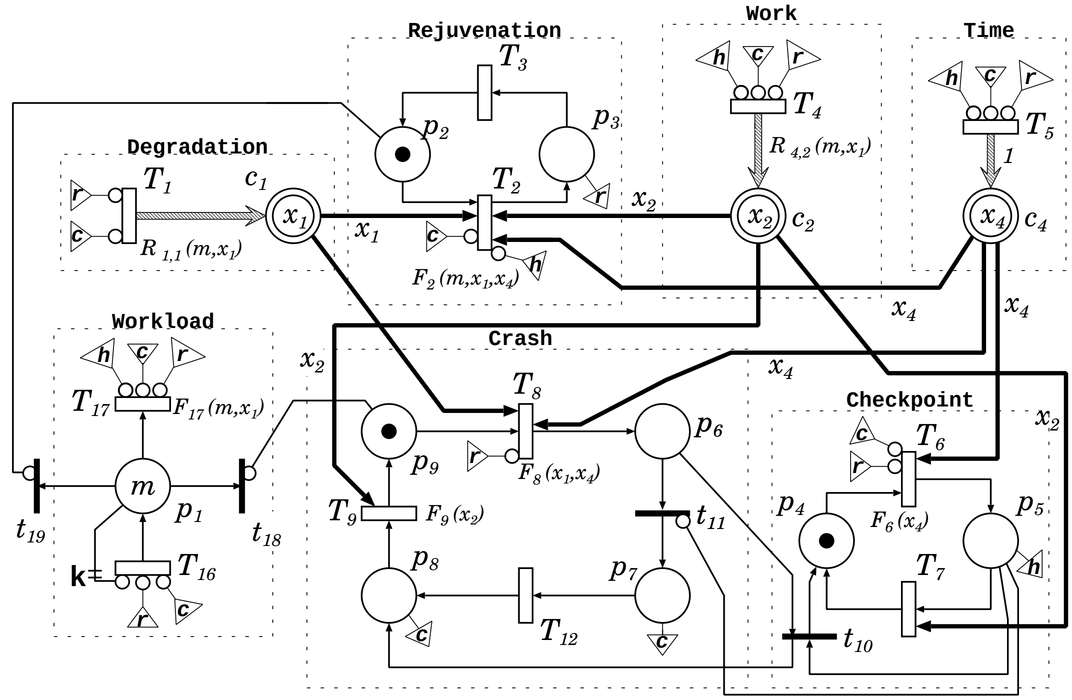

Figure 6: Server Software with checkpointing and rejuvenation

in state $\boldsymbol{m_1}$ all the fluid levels are 0:

$$\pi_1(\tau) = \Pr(\boldsymbol{m}(\tau) = \boldsymbol{m_1});$$

in state $\boldsymbol{m_2}$ the degradation level and the accumulated work may be nonzero:

$$\pi_2(\tau, x_1, x_2) = \lim_{\Delta_1, \Delta_2 \to 0} \frac{\Pr(\boldsymbol{m}(\tau) = \boldsymbol{m_2}, X_1(\tau) \in (x_1, x_1 + \Delta_1), X_2(\tau) \in (x_2, x_2 + \Delta_2))}{\Delta_1 \ \Delta_2};$$

in $\boldsymbol{m_3}$ all the fluid levels are 0:

$$\pi_3(\tau) = \Pr(\boldsymbol{m}(\tau) = \boldsymbol{m_3}).$$

Using the above notations the evolution of the process is described by the following partial differential equations (the equations may be derived using a generalization of the method presented in [24] and [26]):

$$\begin{aligned}
\frac{\partial \pi_0(\tau, x_1, x_2, x_4)}{\partial \tau} + \frac{\partial \pi_0(\tau, x_1, x_2, x_4)}{\partial x_1} + \frac{\partial (\pi_0(\tau, x_1, x_2, x_4) R_{4,2}(x_1))}{\partial x_2} + \frac{\partial \pi_0(\tau, x_1, x_2, x_4)}{\partial x_4} \\
= -\pi_0(\tau, x_1, x_2, x_4)\left[F_2(x_1) + F_6(x_2, x_4) + F_8(x_1)\right] \\
\frac{\partial \pi_1(\tau)}{\partial \tau} = -\lambda \pi_1(\tau) + \int_0^\infty \int_0^\infty \int_0^\infty \pi_0(\tau, x_1, x_2, x_4) F_2(x_1) dx_1 dx_2 dx_4 \\
\frac{\partial \pi_2(\tau, x_1, x_2)}{\partial \tau} + \frac{\partial \pi_2(\tau, x_1, x_2)}{\partial x_1} = -\left(\gamma + F_8(x_1)\right)\pi_2(\tau, x_1, x_2) + \int_0^\infty \pi_0(\tau, x_1, x_2, x_4) F_6(x_2, x_4) dx_4 \\
\frac{\partial \pi_3(\tau)}{\partial \tau} = -\mu \pi_3(\tau) + \int_0^\infty \int_0^\infty \left[\int_0^\infty \pi_0(\tau, x_1, x_2, x_4) dx_4 + \pi_2(\tau, x_1, x_2)\right] F_8(x_1) dx_1 dx_2;
\end{aligned}$$

with initial conditions

$$\begin{aligned}
\pi_0(\tau, 0, 0, 0) &= \lambda \pi_1(\tau) + \mu \pi_3(\tau) \\
\pi_0(\tau, x_1, 0, 0) &= \gamma \int_0^\infty \pi_2(\tau, x_1, x_2) dx_2.
\end{aligned}$$

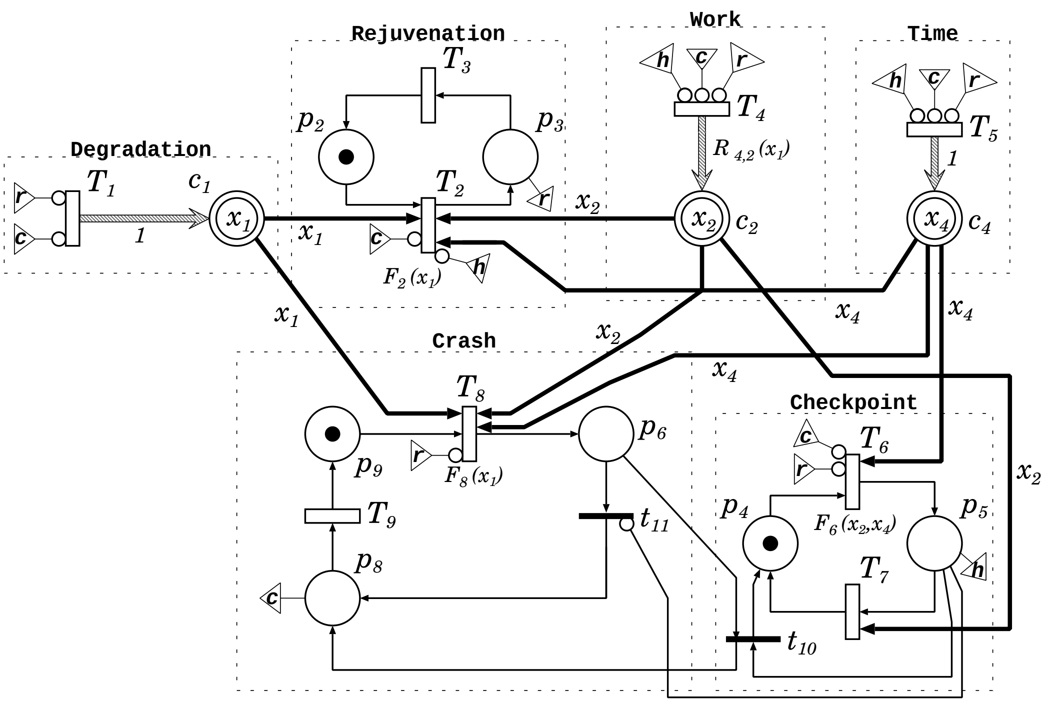

Figure 7: FSPN modeling a system with rejuvenations and checkpoints

In the discrete marking $\boldsymbol{m_0}$ the difference of the fluid levels of c_1 and c_4 (i.e., $x_1 - x_4$) is the degradation level at which the last checkpoint was taken, since $R_{11} = R_{54} = 1$. The last flush out of the fluid of c_2 occurred at that degradation level, and the fluid is filled to c_2 at rate $R_{42}(m, x_1)$ from degradation level $x_1 - x_4$ to x_1, i.e.:

$$x_2 = \int_{x_1 - x_4}^{x_1} R_{4,2}(x)\ dx\ .$$

Hence, in the discrete marking $\boldsymbol{m_0}$, the fluid levels are dependent, and any two of them determine the third one based on this relation. Using the function $x_4 = a(x_1, x_2)$ that determines x_4 based on the fluid levels x_1 and x_2 and the notation $\pi_0^{'}(\tau, x_1, x_2) = \pi_0(\tau, x_1, x_2, a(x_1, x_2))$, the differential equations simplify to the following form:

$$\begin{aligned}
\frac{\partial \pi_0^{'}(\tau, x_1, x_2)}{\partial \tau} + \frac{\partial \pi_0^{'}(\tau, x_1, x_2)}{\partial x_1} &+ \frac{\partial (\pi_0^{'}(\tau, x_1, x_2) R_{4,2}(x_1))}{\partial x_2} \\
&= -\pi_0^{'}(\tau, x_1, x_2)\left[F_2(x_1) + F_6(x_2, a(x_1, x_2)) + F_8(x_1)\right] \\
\frac{\partial \pi_1(\tau)}{\partial \tau} &= -\lambda \pi_1(\tau) + \int_0^\infty \int_0^\infty \pi_0^{'}(\tau, x_1, x_2) F_2(x_1) dx_1 dx_2 \\
\frac{\partial \pi_2(\tau, x_1, x_2)}{\partial \tau} + \frac{\partial \pi_2(\tau, x_1, x_2)}{\partial x_1} &= -\left(\gamma + F_8(x_1)\right) \pi_2(\tau, x_1, x_2) + \pi_0^{'}(\tau, x_1, x_2) F_6(x_2, a(x_1, x_2)) \\
\frac{\partial \pi_3(\tau)}{\partial \tau} &= -\mu \pi_3(\tau) + \int_0^\infty \int_0^\infty \left[\pi_0^{'}(\tau, x_1, x_2) + \pi_2(\tau, x_1, x_2)\right] F_8(x_1) dx_1 dx_2,
\end{aligned}$$

and the initial conditions are

$$\begin{aligned}
\pi_0^{'}(\tau, 0, 0) &= \lambda \pi_1(\tau) + \mu \pi_3(\tau) \\
\pi_0^{'}(\tau, x_1, 0) &= \gamma \int_0^\infty \pi_2(\tau, x_1, x_2) dx_2.
\end{aligned}$$

The probabilities of the discrete markings are obtained by integrating the densities

$$\Pr(\boldsymbol{m}(\tau) = \boldsymbol{m_0}) = \int_0^\infty \int_0^\infty \pi_0^{'}(\tau, x_1, x_2) dx_1 dx_2, \text{ and } \Pr(\boldsymbol{m}(\tau) = \boldsymbol{m_2}) = \int_0^\infty \int_0^\infty \pi_2(\tau, x_1, x_2) dx_1 dx_2.$$

To compute the average rate $C(\tau)$ at which work is checkpointed at time τ we have to take into account both the checkpoints caused by rejuvenation and those occurring independently of a rejuvenation. It may be expressed as:

$$C(\tau) = \int_0^\infty \int_0^\infty x_2 \; (\pi_2(\tau, x_1, x_2) \; \gamma + \pi_0^{'}(\tau, x_1, x_2) \; F_2(x_1)) \; dx_1 dx_2.$$

From which we have that the average checkpointed work until a given time τ is $W_c(\tau) = \int_0^\tau C(\tau) d\tau$. In a similar way, the average rate at which work is lost due to a crash failure is:

$$L(\tau) = \int_0^\infty \int_0^\infty x_2 \; \pi_0^{'}(\tau, x_1, x_2) \; F_8(x_1) dx_1 dx_2,$$

and the average work lost until τ is: $W_l(\tau) = \int_0^\tau L(\tau) d\tau$.

An interesting performance measure of the system which is obtained as a *flow measure* is the ratio of the average checkpointed work and the average work done by the system (i.e., the sum of work checkpointed and lost until time τ). Efficiency may be computed as

$$E(\tau) = \frac{W_c(\tau)}{W_c(\tau) + W_l(\tau)} \tag{2}$$

The set of parameters used in the calculations are the following:

- The work-rate is given by

$$R_{4,2}(x_1) = \begin{cases} r_{max} - (r_{max} - r_{min}) \dfrac{x_1}{\tau_{min}} & x_1 < \tau_{min} \\ r_{min} & x_1 \geq \tau_{min} \end{cases}$$

  So that the work-rate is linearly decreasing until τ_{min}, and after this level of degradation remains constant. In our example $r_{max} = 10$, $r_{min} = 0.5$, $\tau_{min} = 480$.

- The firing rate of the transition T_6 associated with the checkpointing is

$$F_6(x_2, x_4) = \delta(x_2 - \tau_{work}) + \delta(x_4 - \tau_{time}),$$

  where $\delta(x - \tau)$ is the Dirac-impulse at time τ. As a result checkpoint occurs if the level of accumulated work reaches τ_{work} or the time elapsed since the last checkpoint is τ_{time}. The example will be evaluated for different values of τ_{work}. The parameter τ_{time} is equal to 120.

- The firing rate of transition T_2 is $F_2(x_1) = \delta(x_1 - \tau_{rej})$, i.e., rejuvenation is initiated at a degradation level τ_{rej}. The example will be evaluated for different values of τ_{rej}.

- The Weibull hazard function is used for the firing rate $F_8(x_1)$ with shape parameter $\eta = 2$ and scale parameter $\alpha = 2 \times 10^{-6}$. So that the firing rate of T_8 is a linear function of the degradation level $F_8(x_1) = \eta \; \alpha \; x_1$.

- The rates of the exponential transitions are $\lambda = 1/6$, $\gamma = 1$ and $\mu = 1/60$.

Equidistant discretization was applied for the calculations (the time and fluid levels are discretized using the same step size). The correctness of the discretization method was verified by comparing the discretization results with the results given by a simulator. Both the simulator and the discretization algorithm were specifically implemented for this example and not for a general FSPN. The result given by discretization and the simulator are satisfactorily close to each other.

Figures 8, 9, and 10 show the probabilities of the discrete markings over two time scales for the following 3 sets of parameters: `case A`: $\tau_{work} = 200$ and $\tau_{rej} = 200$, `case B`: $\tau_{work} = 200$ and $\tau_{rej} = 400$, `case C`: $\tau_{work} = 400$ and $\tau_{rej} = 400$. The frequent small impulses are associated with checkpointing, while the rare larger impulses represent rejuvenation. The impulses are getting wider and smother as

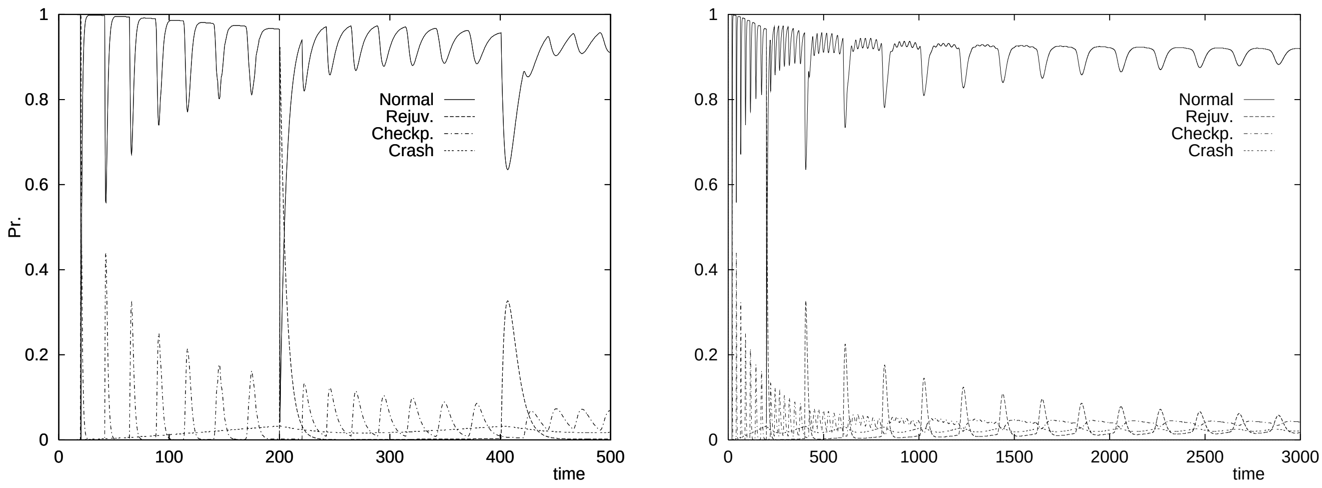

Figure 8: Probabilities of markings in `case A`

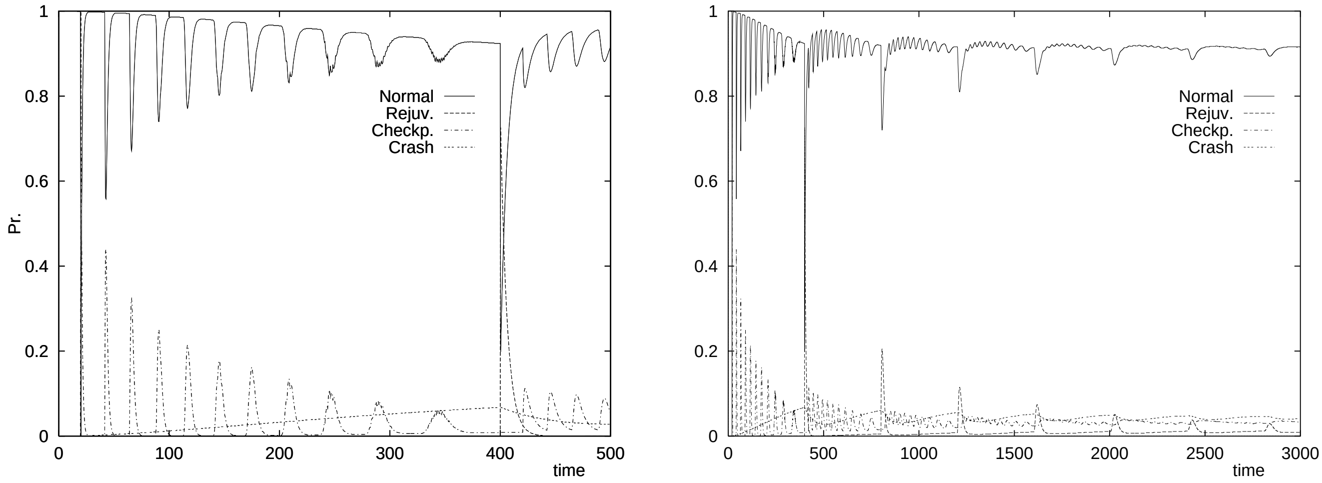

Figure 9: Probabilities of markings in `case B`

time elapses due to the exponentially distributed events while the system undergoes checkpointing and rejuvenation.

Figure 11 shows the efficiency, as defined in (2), of the 3 cases as a function of time. Efficiency is 0 before the first checkpoint by definition. As it can be seen in Figure 11, the efficiency is a performance parameter which does not capture satisfactorily all important consequences of checkpointing and rejuvenation. Based on the definition in (2), the more often the system is checkpointed or rejuvenated the better its efficiency is. The most important shortcoming of the efficiency measure is that it does not consider the time while the system is unavailable. Better performance measures, like the ratio of the (interval average or steady state) performance over the maximum performance level ($R_{42}(0)$) can be evaluated based on the FSPN model. The considered efficiency measure was selected as a simple measure to demonstrate the new features of the proposed FSPN formalism.

6 Conclusions and future works

In this paper, we extend the current FSPN formalism with *flush-out* arcs which enable the fluid in a place to be instantaneously removed. We presented a fairly general Fluid stochastic Petri net, which uses this extension, to model systems with rejuvenation, restoration and checkpointing. We also presented several specialized FSPNs, derived from the general FSPN to model systems with only checkpointing, only rejuvenation and with checkpointing as well as rejuvenation. The last combination, used for server type software systems was not considered before in the literature and our FSPN model

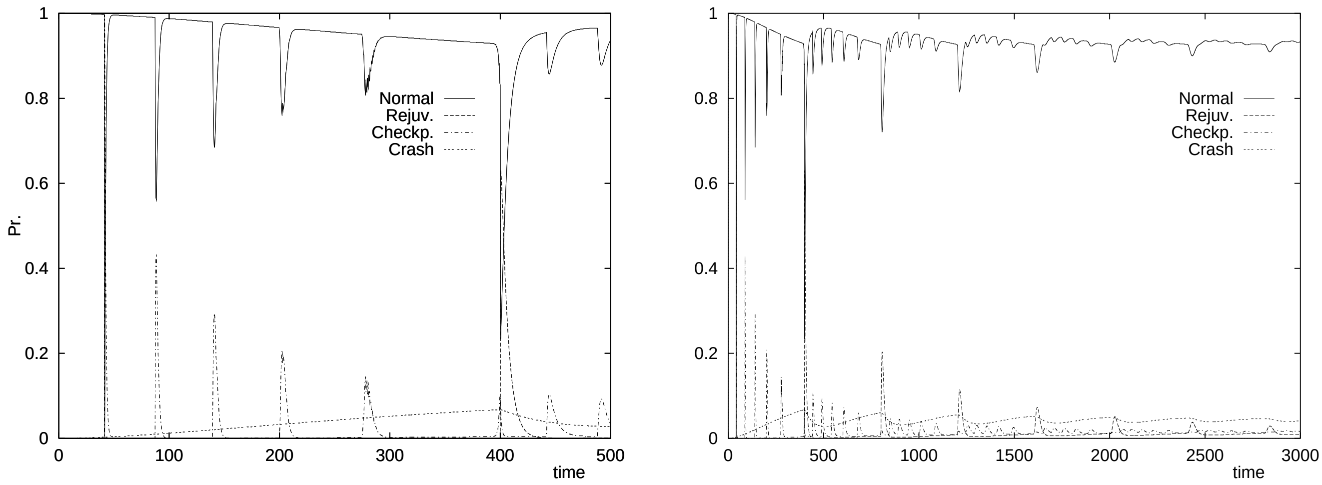

Figure 10: Probabilities of markings in `case C`

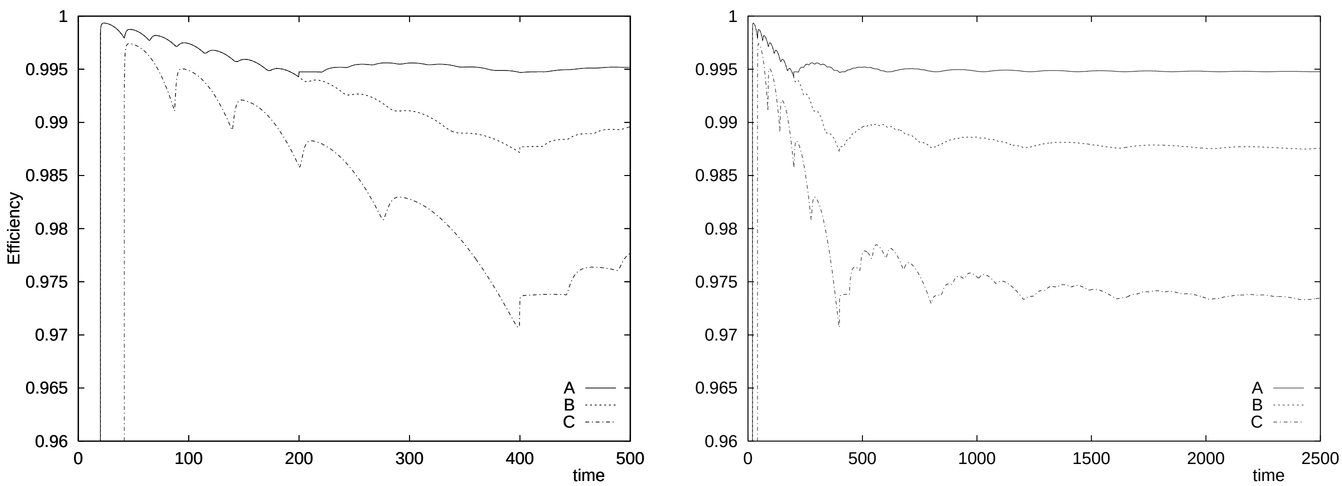

Figure 11: Efficiency

constitutes a novel contribution. The intent in presenting several special models was twofold. First we wanted to show that it is indeed possible to capture the effects of checkpointing, restoration and rejuvenation as well as any combination of one or more schemes using the extended FSPN formalism. Second we wanted to show that models previously reported in the literature for such systems can be cast in the proposed FSPN framework. Moreover, some of the assumptions made in these models can be generalized and still be captured in our FSPN framework. We also showed that the FSPN formalism is effective, in the sense that it can be solved using numerical and/or simulation techniques to obtain state probabilities and to derive measures of interest.

However, in order to make FSPNs to be a viable modeling formalism, several issues remain to be addressed. First, extensions which allow discrete, random jumps in the fluid level of a continuous place need to be incorporated in addition to the jump-to-zero of the fluid level, provided by the *flush-out* arcs. This would raise issues about the stability of the underlying stochastic process. Therefore, for the extended formalism to be theoretically sound, precise necessary/sufficiency conditions for stability need to be determined. An orthogonal need is in the availability of robust software tools which can solve an FSPN model via numerical or simulation techniques.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[2] H. Alla and R. David. Autonomous and Timed Continuous Petri Nets. *Lecture Notes in Computer Science; Advances in Petri Nets 1993*, 674:71–90, 1993.

[3] H. Alla and R. David. Continuous and Hybrid Petri Nets. *Journal of Systems Circuits and Computers*, 8(1), Feb 1998. Special issue on Petri Nets.

[4] A. Bobbio, A. Puliafito, M. Telek, and K. S. Trivedi. Recent Developments in Non-Markovian Stochastic Petri Nets. *Journal of Systems Circuits and Computers*, 8(1):119–158, Feb 1998.

[5] A. Bobbio and M. Sereno. Fine Grained Software Rejuvenation Models. In *Proc. 3-th International Computer Performance & Dependability Symposium (IPDS '98)*, pages 4–12, Durham, North Carolina, USA, September 1998. IEEE Comp. Soc. Press.

[6] R. V. Campos and E. de Sousa e Silva. Availability and performance evaluation of database systems under periodic checkpoints. In *Proc. of the 25th IEEE Intnl. Symposium on Fault Tolerant Computing (FTCS)*, pages 269–277, Pasadena, California, 1995.

[7] P. F. Chimento and K. S. Trivedi. The Completion Time of Programs on Processors Subject to Failure and Repair. *IEEE Transactions on Computers*, 42(10):1184–1194, October 1993.

[8] G. Ciardo, J. K. Muppala, and K. S. Trivedi. On the solution of GSPN reward models. *Performance Evaluation*, 12(4):237–253, 1991.

[9] G. Ciardo, D. M. Nicol, and K. S. Trivedi. Discrete-event Simulation of Fluid Stochastic Petri Nets. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 217–225, Saint Malo, France, June 1997. IEEE Comp. Soc. Press.

[10] E. G. Coffman and E. N. Gilbert. Optimal strategies for scheduling checkpoints and preventive maintenance. *IEEE Transactions on Reliability*, 39(1):9–18, April 1990.

[11] A. Duda. The effects of checkpointing on program execution time. *Information Processing Letters*, 16:221–229, 1983.

[12] A. I. Elwalid and D. Mitra. Statistical Multiplexing with Loss Priorities in Rate-Based Congestion Control of High-Speed Networks. *IEEE Transaction on Communications*, 42(11):2989–3002, November 1994.

[13] S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi. Minimizing completion time of a program by checkpoint and rejuvenation. In *Proc. 1996 ACM SIGMETRICS Conference*, pages 252–261, Philadelphia, PA, May 1996.

[14] S. Garg, A. Pfening, A. Puliafito, M. Telek, and K. S. Trivedi. Modeling and analysis of load and time dependent software rejuvenation policies. In *Proceedings 3-rd International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS3)*, pages 35–39, Bloomingdale, IL, Sep 1996.

[15] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of software rejuvenation using Markov regenerative stochastic Petri net. In *Proc. of* 6^{th} *Int. Symposium on Software Reliability Engineering (ISSRE95)*, Toulouse, France, October 1995.

[16] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. On the analysis of software rejuvenation policies. In *Proc. 12th Annual Conference on Computer Assurance (COMPASS)*, Gaitersburg, MD, June 16-20 1997.

[17] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of preventive maintenance in transaction based software systems. *IEEE Trans. on Computers*, 47(1):96–107, 1998. Special issue on Dependability of Computing Systems.

[18] S. Garg, A. van Moorsel, K. S. Trivedi, and K. Vaidyanathan. A methodology for detection and estimation of software aging. In *Proc. of the Ninth International Symposium on Software Reliability Engineering*, pages 282–292, Paderborn, Germany, November 4-7 1998.

[19] E. Gelenbe. On the optimum checkpointing interval. *Journal of ACM*, 2(27):259–270, April 1979.

[20] E. Gelenbe and M. Hernandez. Optimum checkpoints with age dependent failures. *Acta Informatica*, 27:519–531, 1990.

[21] J. Gray. Why do computers stop and what can be done about it? In *Proc. of 5th Symp. on Reliability in Distributed Software and Database Systems*, pages 3–12, January 1986.

[22] J. Gray and D. P. Siewiorek. High-availability computer systems. *IEEE Computer*, pages 39–48, September 1991.

[23] B. O. A. Grey. Making SDI software reliable through fault-tolerant techniques. *Defense Electronics*, pages 77–80,85–86, August 1987.

[24] G. Horton, V. G. Kulkarni, D. M. Nicol, and K. S. Trivedi. Fluid stochastic Petri Nets: Theory, Application, and Solution Techniques. *European Journal of Operations Research*, 105(1):184–201, Feb 1998.

[25] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton. Software rejuvenation: Analysis, module and applications. In *Proc. of* 25^{th} *Int. Symposium on Fault-Tolerance Computing (FTCS-25)*, Pasadena, CA, USA, June 1995.

[26] R. J. Karandikar and V. G. Kulkarni. Second-Order Fluid Flow Models: Reflected Brownian Motion in a Random Environment. *Operations Research*, 43(1):77–88, 1995.

[27] V. G. Kulkarni, V. F. Nicola, and K. S. Trivedi. The completion time of a job on multi-mode systems. *Advances in Applied Probability*, 19:491–496, December 1987.

[28] V. G. Kulkarni, V. F. Nicola, and K. S. Trivedi. Effects of checkpointing and queuing on program performance. *Communications on Statistics- Stochastic Models*, 6(4):615–648, 1990.

[29] P. Lecuyer and J. Malenfant. Computing optimal checkpointing strategies for rollback and recovery systems. *IEEE Transactions on Computers*, C-37(4):491–496, April 1988.

[30] E. Marshall. Fatal error: how Patriot overlooked a Scud. *Science*, 13:1347, March 1992.

[31] A. Pfening, S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Optimal rejuvenation for tolerating soft failures. *Performance Evaluation*, 27 & 28:491–506, October 1996.

[32] K. G. Shin, T. Lin, and Y. Lee. Optimal checkpointing of real-time tasks. *IEEE Transactions on Computers*, C-36(11), November 1987.

[33] M. Sullivan and R. Chillarege. Software defects and their impact on system availability - a study of field failures in operating systems. In *Proc. 21st IEEE Intnl. Symposium on Fault-Tolerant Computing*, pages 2–9, 1991.

[34] A. T. Tai, S. N. Chau, L. Alkalaj, and H. Hecht. On-board preventive maintenace: analysis of effectiveness and optimal duty period. In *Proc. of 3rd Intnl. Worskshop on Object-oriented Real-time Dependable Systems*, Newport Beach, California, February 1997.

[35] K. Trivedi and V. Kulkarni. FSPNs: Fluid Stochastic Petri nets. In *Application and Theory of Petri Nets 1993, Proc.* 14^{th} *Intern. Conference*, LNCS, Chicago, USA, June 1993. Springer Verlag.

[36] K. Wolter. Second order fluid stochastic petri nets: an extension of gspns for approximate and continuous modelling. In *Proc. of World Congress on System Simulation*, pages 328–332, Singapore, Sep 1997.