On the Performance Estimation and Resource Optimization in Process Petri Nets

Ricardo J. Rodríguez, Jorge Júlvez, and José Merseguer

***Abstract*—Many artificial systems can be modeled as discrete dynamic systems in which resources are shared among different tasks. The performance of such systems, which is usually a system requirement, heavily relies on the number and distribution of such resources. The goal of this paper is twofold: first, to design a technique to estimate the steady-state performance of a given system with shared resources, and second, to propose a heuristic strategy to distribute shared resources so that the system performance is enhanced as much as possible. The systems under consideration are assumed to be large systems, such as service-oriented architecture (SOA) systems, and modeled by a particular class of Petri nets (PNs) called process PNs. In order to avoid the state explosion problem inherent to discrete models, the proposed techniques make intensive use of linear programming (LP) problems.**

***Index Terms*—Discrete event systems (DESs), performance evaluation, Petri nets (PNs), software performance.**

I. Introduction

NOWADAYS, the majority of systems in several domains (such as manufacturing, logistics, or web services) are complex systems using shared resources. Usually, the number of resources is the key for the system to obtain a good throughput (defined as jobs completed per unit of time) for a large number of users/clients. However, the number of resources (e.g., the number of servers) cannot be always increased as desired. In the real world, each project of a new system manages a budget, and this budget limits the number of resources that can be acquired.

Many artificial systems can be naturally modeled as discrete event systems (DESs). Unfortunately, these systems are usually large, and this makes the exact computation of their performance a highly complex computational task. The main reason for this complexity is the well-known state explosion problem. As a result, a task that requires an exhaustive state-space exploration becomes unachievable in a reasonable time for large systems.

The framework of this paper is that of a DES dealing with the resource-allocation problem, also called resource-allocation systems (RASs) [1], modeled with Petri nets (PNs); more precisely, we will focus on process PNs [2]. A large number of works in the literature deal with RASs from a qualitative point of view (computing deadlock avoidance [3]–[7] or siphon structures [8], [9]), whereas our vision here is different: We focus on the quantitative point of view. In particular, the goals of this paper are: 1) to efficiently estimate the throughput of a system; and 2) to find a near-optimal distribution of resources for the so-called process PNs. To the best of our knowledge, this resource optimization issue has not been studied in the research community for process PNs.

To fulfill these goals, in this paper, we propose in the first place an iterative strategy to compute upper throughput bounds closer to the real throughput[1] than those achieved in previous works [10], [11]. In the second place, we propose a heuristic iterative strategy to gauge in the best possible way the number of resources needed so that the overall system throughput is maximized. Both strategies use linear programming (LP) techniques for which polynomial complexity algorithms exist, thus offering a good tradeoff between accuracy and computational complexity.

Let us summarize how the strategies presented here work. The strategy for obtaining sharper (i.e., closer to the real throughput) upper throughput bounds is based on the computation of *bottlenecks*. It calculates in the first step the slowest part of the system, i.e., the initial bottleneck of the system. After that, in each iteration, the most likely part of the system to be constraining the current bottleneck is calculated, and the union of both parts is considered to calculate the new upper throughput bound. The heuristic strategy for resource optimization is aimed at calculating the number of resources required for the bottleneck in order not to constraint the system throughput.

Both strategies can be applied to any real-life application whose PN model matches the net class considered in this paper, i.e., process PNs. This kind of real-life application can be found in manufacturing, logistics, or other systems, such as web services. In general, such applications represent real-life problems where resources are shared.

Running example. Let us consider a simple supermarket, where customers arrive and look for the products that they want to buy. After spending some time in the supermarket, the customer wants to pay for the products and goes to the checkout. The customer may choose to pay either in cash (with probability $p \in [0, \ldots, 1]$) or by a credit card (with probability $1 - p$). In the latter case, the cashier will need a point-of-sales

Manuscript received June 14, 2011; revised December 7, 2011, April 17, 2012, and September 21, 2012; accepted December 15, 2012. This work was supported in part by the Spanish Ministry of Science under Project DPI2010-20413. This paper was recommended by Associate Editor W. van der Aalst.

The authors are with the Department of Computer Science and Systems Engineering, University of Zaragoza, 50018 Zaragoza, Spain (e-mail: rjrodriguez@unizar.es; julvez@unizar.es; jmerse@unizar.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2013.2245118

[1]The notion of real throughput refers to the throughput of the system modeled, which can be calculated by exact analysis or simulation.

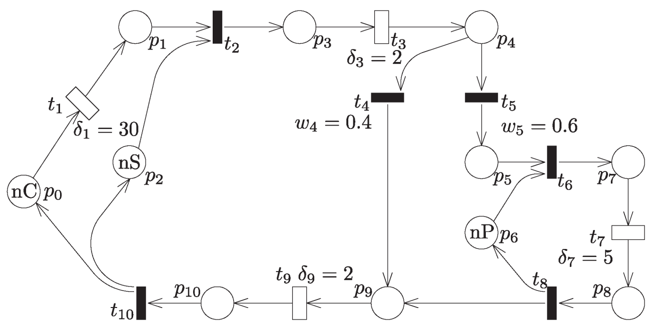


Fig. 1. Example of a supermarket system.

(PoS) terminal to complete the payment. Fig. 1 depicts a PN modeling of the supermarket system. The PN represents the number nC of clients (initial marking of place p_0) and the number nS of cashiers (initial marking of place p_2) who attend to the customers and the PoS terminals, represented by the initial marking nP of place p_6. Immediate transitions are represented by a black box, whereas exponential transitions are depicted by a white box. The think time of customers is represented by transition t_1, which follows an exponential distribution with mean $\delta_1 = 30$ min, whereas transition t_3 represents the time for attending to customers, which follows an exponential distribution with mean $\delta_3 = 2$ min. The choice of the mode of payment is represented by place p_4. A payment in cash occurs with probability $w_4 = 0.4$, whereas credit card payment has probability $w_5 = 0.6$. The use of the PoS terminals (represented by transition t_7) takes, in terms of time, about 5 min, i.e., $\delta_7 = 5$, Finally, the cashier spends, on average, $\delta_7 = 2$ min on completing the customer transaction, which is represented by transition t_9.

With the above PN configuration, it is interesting to know, for example, where the bottleneck of the system is. Is the slowest part of the system the cashier's work or is it the PoS terminal? Another question of interest is whether the system's resources are sufficient to attend to an expected number of customers. Suppose that there is a budgetary limitation on the supermarket's expenditure and that the cost of hiring new cashiers and buying new PoS terminals is known. Where and in what ratio should the money be spent? These are the kinds of questions dealt in this paper.

Suppose an initial marking of $nC = 5$ expected customers, $nS = 2$ cashiers, and $nP = 2$ PoS terminals. With this initial configuration, no matter how many new cashiers are hired or how many new PoS terminals are bought, the resources impose no constraint on the system. There are enough resources to attend to this number of customers with a think time of $\delta_1 = 30$ min. Nevertheless, if the number of expected customers is set to $nC = 100$, and the same think time is considered, the bottleneck in the system will be in the number of cashiers. This indicates that more cashiers should be hired if it is desired to attend to customers with the same think time.

This paper is organized as follows. Section II discusses related work. In Section III, some basic concepts are introduced, such as the kind of PNs that we are dealing with. In Section IV, a new iterative algorithm for performance estimation is presented, whereas a new resource optimization technique is explained in Section V. Section VI introduces a case study to demonstrate our methods and describes the experiments carried out, together with our conclusions. Finally, Section VII summarizes the main contributions of this paper.

II. Related Work

Performance estimation using PNs has been extensively studied. Some works are concerned with the exact computation of analytical measures of performance [12], whereas others overcome the state explosion problem providing performance bounds [10], [11], [13]–[15]. The use of performance bounds, on which our approach is based, avoids the necessity of calculating the whole state space. The advantage of using performance bound computation is the reduced computing time, but its drawback is the difficulty of assessing how accurate the computed bound is with respect to the real system performance.

One of the first works on performance bound computation is [13], where strongly connected marked graphs (MGs) with deterministic timing are considered, and the reachability of the computation bound is proven. Some other works that compute performance bounds use LP techniques [10], [11] in the same way as in our approach. These bounds are frequently calculated by using the first-order moment (i.e., the mean) of the distributions associated to the firing delay. Such bounds were improved in [14] for the particular case of MGs by using regrowing techniques (i.e., by adding more components to the initial bottleneck of the net). In [15], the second-order moment is used to obtain a sharper (i.e., more accurate) performance bound.

Other works provide bounds for queueing systems instead of PN models, e.g., [16]–[18]. In [16], Haddad *et al.* give space complexity upper and lower bounds for Stochastic PNs with a product-form solution. In [17], Casale *et al.* propose performance upper and lower bounds for closed queueing networks with general independent and nonrenewal services. They use LP techniques on the queue activity probabilities. In [18], Osogami and Raymond provide upper and lower bounds on the tail distribution of the transient waiting time for a general independent service queue. They use the two first moments of the service time and interarrival time and compute the bounds through semidefinite programming, which is a convex optimization technique used for optimization of complex systems. In contrast, our approach uses first-order moment and LP techniques.

Resource optimization and its usage have already been studied for workflow PNs (WF-nets) [19] or some variants [20]–[22]. The underlying PN model of WF-nets is free-choice nets (FCNs). However, the kind of systems that we are considering cannot be modeled through FCNs. In the systems that we consider, there may be conflicts in the resource acquisition synchronization, which is not allowed in FCNs. Li *et al.* propose in [19] an approach to estimate the resource availability by using continuous-time Markov chains (CTMCs) and compute the turnaround time (i.e., the shortest response time) by performing reduction operations on the original WF-net. This performance analysis has exponential complexity in the worst case, whereas our approach has polynomial complexity due to the use of LP techniques. Resource usage could be computed in our approach by calculating the average marking of resource places in the PN system. In [20], Wang and Zeng provide a method for computing the best implementation case for a workflow represented by a PN model, based on the reachability graph. Such a method, however, can suffer scalability problems if the workflow size is large. In [21], Hee *et al.* provide an algorithm to compute optimal resource allocation in stochastic WF-nets. Such an algorithm suffers from scalability problems because its complexity depends on the number of resources. In contrast, our approach only depends on the net structure, irrespective of the number of resources in the system. Therefore, for large systems with a great number of resources, our approach is more tractable than that set out in [21]. Chen *et al.* propose in [22] a new PN model, called the Resource Assignment PN, to define how resources are shared and assigned among different and concurrent project activities. The computation of the execution project time considers deterministic timing, and unlike our approach, this new PN model is not able to model activities acquiring and releasing resources intermittently.

Another important issue related to resource sharing is deadlock prevention. The common use of system resources in concurrent systems may lead to deadlock problems, i.e., a process waits for the evolution of another process or processes, whereas the latter also wait for the former to evolve. In order to deal with such problems, there exist deadlock prevention or avoidance policies that may be applied for assuring the liveness property, thus avoiding deadlocks [3]–[7], [23], [24]. As this issue has been extensively studied in the literature (a recently published review can be found in [7]) and is not the main focus of this paper, we assume that all the PNs considered here are live.

With respect to the aforementioned works, the contributions of this paper are the following. First, we provide a method to compute upper throughput bounds with greater accuracy than the upper bounds achieved in the aforesaid works. Then, we provide a heuristic iterative strategy to distribute, for a given budget, the number of resources in the best possible way so that the overall system throughput is maximized.

III. Preliminary Concepts and Definitions

Some basic concepts are introduced here regarding the special class of PNs that we are considering and its main characteristics. First, we define PNs in the untimed framework and the process PN formalism. Subsequently, we define timed PN systems (visit ratios, average marking, and steady-state throughput). In what follows, the reader is assumed to be familiar with PNs (see [25] for a gentle introduction).

A. Untimed PNs

Definition 1: A PN [25] is a 4-tuple $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$, where:

- P and T are disjoint nonempty sets of *places* and *transitions* ($|P| = n,\ |T| = m$);
- $\mathbf{Pre}$ ($\mathbf{Post}$) are the preincidence (postincidence) nonnegative-integer matrices of size $|P| \times |T|$.

The preset and postset of a node $v \in P \cup T$ are defined as ${}^\bullet v = \{u \in P \cup T | (u, v) \in F\}$ and $v^\bullet = \{u \in P \cup T | (v, u) \in F\}$, respectively, where $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs. A PN is said to be *self-loop free* if, $\forall p \in P$, $t \in T t \in {}^\bullet p$ implies $t \notin p^\bullet$. *Ordinary* nets are PNs whose arcs have weight 1. The *incidence matrix* of a PN is defined as $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$.

A vector $\mathbf{m} \in \mathbb{Z}^{|P|}_{\geq 0}$ that assigns a nonnegative integer to each place is called a *marking vector* or *marking*.

Definition 2: A *PN system* or *marked PN* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0}\rangle$ is PN $\mathcal{N}$ with *initial marking* $\mathbf{m_0}$.

The set of markings *reachable* from $\mathbf{m_0}$ in $\mathcal{N}$ is denoted as $\mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and is called the *reachability set*.

A place $p \in P$ is *k-bounded* if and only if, $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}),\ \mathbf{m}(p) \leq k$. Net system $\mathcal{S}$ is *k-bounded* if and only if each place is k-bounded. A net system is *bounded* if and only if there exists some k for which it is k-bounded. Net $\mathcal{N}$ is *structurally bounded* if and only if it is bounded no matter which $\mathbf{m_0}$ is the initial marking.

A transition $t \in T$ is *enabled* at marking $\mathbf{m}$ if $\mathbf{m} \geq \mathbf{Pre}(\cdot, t)$, where $\mathbf{Pre}(\cdot, t)$ is the column of $\mathbf{Pre}$ corresponding to transition t. A transition t enabled at $\mathbf{m}$ can *fire,* yielding a new marking $\mathbf{m}' = \mathbf{m} + \mathbf{C}(\cdot, t)$ (*reached* marking). This is denoted by $\mathbf{m} \xrightarrow{t} \mathbf{m}'$. A sequence of transitions $\sigma = \{t_i\}^n_{i=1}$ is a *firing sequence* in $\mathcal{S}$ if there exists a sequence of markings, such that $\mathbf{m_0} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \mathbf{m}_2 \cdots \xrightarrow{t_n} \mathbf{m}_n$. In this case, marking $\mathbf{m}_n$ is said to be *reachable* from $\mathbf{m_0}$ by firing σ, and this is denoted by $\mathbf{m}_0 \xrightarrow{\sigma} \mathbf{m}_n$. The *firing count vector* $\boldsymbol{\sigma} \in \mathbb{Z}^{|T|}_{\geq 0}$ of the firing sequence σ is a vector, such that $\boldsymbol{\sigma}(t)$ represents the number of occurrences of $t \in T$ in σ. If $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, then we can write in vector form $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$, which is referred to as the *linear* (or *fundamental*) *state equation* of the net.

Two transitions, i.e., t and t', are said to be in *structural conflict* if they share, at least, one input place, i.e., ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Two transitions t and t' are said to be in *effective conflict for a marking* $\mathbf{m}$ if they are in structural conflict and if they are both enabled at $\mathbf{m}$. Two transitions t and t' are in *equal conflict* if $\mathbf{Pre}(\cdot, t) = \mathbf{Pre}(\cdot, t') \neq \mathbf{0}$, where $\mathbf{0}$ is a vector with all entries equal to zero.

A transition t is *live* if it can be fired from every reachable marking. A transition t is *dead* for a reachable marking $\mathbf{m}$ if and only if $\forall \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m}),\ \neg(\mathbf{m} \xrightarrow{t} \mathbf{m}')$. A marked PN $\mathcal{S}$ is *live* when every transition is live.

A *p-semiflow* is a nonnegative-integer vector $\mathbf{y} \geq \mathbf{0}$, such that it is a left annuler of the net's incidence matrix, i.e., $\mathbf{y} \cdot \mathbf{C} = 0$ (in the sequel, we omit the transpose symbol in the matrices and vectors for clarity). A p-semiflow implies a token conservation law independent from any firing of transitions. A *t-semiflow* is a nonnegative-integer vector $\mathbf{x} \geq \mathbf{0}$, such that it is a right annuler of the net's incidence matrix, i.e., $\mathbf{C} \cdot \mathbf{x} = 0$. A p-semiflow (or t-semiflow) $\mathbf{v}$ is *minimal* when its support, i.e., $\|\mathbf{v}\| = \{i | \mathbf{v}(i) \neq 0\}$, is not a proper superset of the support of any other p-semiflow (or t-semiflow), and the greatest common divisor of its elements is 1. For example, the PN depicted in Fig. 1 has three minimal p-semiflows: $\|\mathbf{y}_1\| = \{p_0, p_1, p_3, p_4, p_5, p_7, p_8, p_9, p_{10}\}$, $\|\mathbf{y}_2\| = \{p_2, p_3, p_4, p_5, p_7, p_8, p_9, p_{10}\}$, and $\|\mathbf{y}_3\| = \{p_6, p_7, p_8\}$. A PN is said to be *conservative* (*consistent*) if there exists a p-semiflow (t-semiflow), which contains all places (transitions) in its support.

A PN is said to be *strongly connected* if there is a directed path joining any pair of nodes of the graph. A *state machine* is a particular type of an ordinary PN, where each transition has exactly one input arc and exactly one output arc, i.e., $|t^\bullet| = |{}^\bullet t| = 1\ \forall t \in T$. In this paper, we focus on process PNs, which are defined as follows [2]:

Definition 3: A process PN is a strongly connected self-loop-free PN $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$, with the following characteristics.

1) $P = P_0 \cup P_S \cup P_R$ is a partition, such that $P_0 = \{p_0\}$ is the *process-idle place* $P_S \neq \emptyset, P_S \cap P_0 = \emptyset, P_S \cap P_R = \emptyset$; P_S is the set of *process-activity places*; and $P_R = \{r_1, \ldots, r_n\},\ n > 0, P_R \cap P_0 = \emptyset$ is the set of *resources places*.
2) Subnet $\mathcal{N}' = \langle P \setminus P_R, T, \mathbf{Pre}, \mathbf{Post}\rangle$ is a strongly connected state machine, such that every cycle contains p_0.
3) For each $r \in P_R$, there exists a unique minimal p-semiflow associated to r, $\mathbf{y}_r \in \mathbb{N}^{|P|}$, fulfilling $\|\mathbf{y}_r\| \cap P_R = \{r\}, \|\mathbf{y}_r\| \cap P_S \neq \emptyset, \|\mathbf{y}_r\| \cap P_0 = \emptyset$ and $\mathbf{y}_r(r) = 1$. This establishes how each resource is reused, i.e., the resources cannot be created or destroyed.
4) $P_S = \bigcup_{r \in P_R}(\|\mathbf{y}_r\| \setminus \{r\})$.

Definition 3 implies that process PNs are conservative and consistent.

Let $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$ be a process PN. A vector $\mathbf{m_0} \in \mathbb{Z}^{|P|}_{\geq 0}$ is called *acceptable initial marking* [2] of $\mathcal{N}$ if and only if: 1) $\mathbf{m_0}(p) \geq 1,\ p \in P_0$; 2) $\mathbf{m_0}(p) = 0,\ \forall p \in P_S$; and 3) $\mathbf{m_0}(r) \geq \mathbf{y}_r(r),\ \forall r \in P_R$, where $\mathbf{m_0}(r)$ is the *capacity* of the resource r, and $\mathbf{y}_r$ is the unique minimal p-semiflow associated to r.

Definition 4: A *process PN system* or *marked process PN* $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0}\rangle$ is a process PN $\mathcal{N}$ with an *acceptable initial marking* $\mathbf{m_0}$.

In this paper, we assume that the first acquired resource in the process PNs under study is a resource that represents the maximum capacity of the process, its capacity always being greater than the number of instances in the process-idle place. Therefore, such a resource place becomes implicit, and we do not consider it for the analysis.

B. Timed PNs

Definition 5: A timed process PN (TPPN) system is a tuple $\langle \mathcal{S}, \mathbf{s}, \mathbf{r}\rangle$, where $\mathcal{S}$ is a process PN system, $\mathbf{s} \in \mathbb{R}^{|T|}_{\geq 0}$ is the vector of average service times of transitions, and $\mathbf{r} \in \mathbb{N}^{|T|}_{>0}$ is the vector of rates associated to transitions.

If $\mathbf{s}(t) > 0$, then transition t is a timed transition. Otherwise, i.e., $\mathbf{s}(t) = 0$, transition t is immediate. It will be assumed that all transitions in conflict are immediate. Immediate transition t in conflict will fire with probability $\mathbf{r}(t)/(\sum_{t' \in A} \mathbf{r}(t'))$, where A is the set of enabled immediate transitions in conflict. The firing of immediate transitions consumes no time. When a timed transition becomes enabled, it fires following an exponential distribution with mean $\mathbf{s}(t)$. There exist different semantics for the firing of transitions, *infinite* and *finite* server semantics being the most frequently used. In this paper, we will assume that the timed transitions work under infinite server semantics.

The average marking vector $\overline{\mathbf{m}}$ in an ergodic PN system is defined as [26]

$$\overline{\mathbf{m}}(p) \underset{\text{AS}}{=} \lim_{\tau \to \infty} \frac{1}{\tau} \int_0^\tau \mathbf{m}(p)_u du \tag{1}$$

where $\mathbf{m}(p)_u$ is the marking of place p at time u and the notation $\underset{\text{AS}}{=}$ means *equal almost surely*.

Similarly, the steady-state throughput χ in an ergodic PN is defined as [26]

$$\chi(t) \underset{\text{AS}}{=} \lim_{\tau \to \infty} \frac{\boldsymbol{\sigma}(t)_\tau}{\tau} \tag{2}$$

where $\boldsymbol{\sigma}(t)_\tau$ is the firing count of transition t at time τ.

By definition, all the places of a TPPN are covered by p-semiflows; therefore, it is structurally bounded. In this paper, we will assume that the TPPN under study is a live and structurally bounded net with freely related t-semiflows (i.e., an FRT-net) [27]. The range of nets fulfilling these conditions are relatively broad. Examples of TPPNs that are FRT-nets are: TPPNs in which $\mathcal{N}'$ is choice free, and TPPNs in which $\mathcal{N}'$ satisfies that every conflict is an equal conflict. It is known that the CTMC associated to these nets is ergodic [27], which implies the existence of the given limits.

The vector of visit ratios expresses the relative throughput of transitions in the steady state. The visit ratio $\mathbf{v}(t)$ of each transition $t \in T$ normalized for transition t_i, i.e., $\mathbf{v}^{t_i}(t)$, is expressed as follows:

$$\mathbf{v}^{t_i}(t) = \frac{\chi(t)}{\chi(t_i)} = \boldsymbol{\Gamma}(t_i) \cdot \chi(t) \qquad \forall t \in T \tag{3}$$

where $\boldsymbol{\Gamma}(t_i) = 1/(\chi(t_i))$ represents the *average interfiring time* of transition t_i.

In FRT-nets, the vector of visit ratios $\mathbf{v}$ exclusively depends on the structure of the net and on the routing rates [27]. Thus, the vector of visit ratios $\mathbf{v}$ normalized for transition t_i, i.e., $\mathbf{v}^{t_i}$, can be calculated by solving the following linear system of equations [27]:

$$\begin{pmatrix} \mathbf{C} \\ \mathbf{R} \end{pmatrix} \cdot \mathbf{v}^{t_i} = 0 \quad \mathbf{v}^{t_i}(t_i) = 1 \tag{4}$$

where $\mathbf{R}$ is a matrix containing rates $\mathbf{r}(t)$ associated to transitions in equal conflict.

IV. Performance Estimation

Here, we present a new iterative algorithm to compute upper throughput bounds of a TPPN system. Such an algorithm is based on the computation of p-semiflows. Each p-semiflow has an associated subnet composed of the places in the support of the p-semiflow. Given that such a subnet satisfies a conservation law and synchronizes with other subnets in the overall system, its throughput, if the subnet is considered isolated, imposes an upper throughput bound for the overall system. The proposed iterative strategy considers initially the p-semiflow with lowest throughput. Its associated subnet is called the initial bottleneck.

The bottleneck is increased by adding the associated subnet to the subnet associated to the next most constraining p-semiflow.

A. Little's Law and Upper Bounds

The Little's formula [28] conditions involve the average number of customers L in the system, throughput λ, and the average time W spent by a customer within the system, i.e.,

$$L = \lambda \cdot W. \tag{5}$$

Let p be a place such that $|p^\bullet| = 1$ and $p^\bullet = \{t\}$; then, pair (p, t) can be seen as a simple queueing system to which, if the limits of *average marking* and *steady-state throughput* exist, Little's formula can be directly applied [27] as

$$\overline{\mathbf{m}}(p) = (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \mathbf{r}(p) \tag{6}$$

where $\mathbf{Pre}(p,t) \cdot \chi(t)$ is the output rate of tokens from place p, which in steady state is equal to the input rate, and $\mathbf{r}(p)$ is the average residence time at place p, i.e., the average time spent by a token in place p.

The average residence time $\mathbf{r}(p)$ is the sum of the average waiting time due to a possible synchronization and the average service time $\mathbf{s}(t)$. Therefore, (6) becomes

$$\overline{\mathbf{m}}(p) = (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \mathbf{r}(p) \geq (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \mathbf{s}(t) \tag{7}$$

where the service time $\mathbf{s}(t)$ is a lower bound for the average residence time $\mathbf{r}(p)$, i.e., $\mathbf{s}(t) \leq \mathbf{r}(p)$, since place p has only one output transition. Given that conflicting transitions are assumed to be immediate, (7) can also be applied to any pair (p, t), with $t \in p^\bullet$ and with t being a transition in conflict. Hence, the following system of inequalities can be derived [27] from (3) and (7) as follows:

$$\mathbf{\Gamma}(t_i) \cdot \overline{\mathbf{m}} \geq \mathbf{Pre} \cdot \mathbf{D}^{\mathbf{t_i}} \tag{8}$$

where $\mathbf{\Gamma}(t_i)$ is the average interfiring time of transition t_i, and $\mathbf{D}^{\mathbf{t_i}}$ is the vector of *average service demands of transitions*, i.e., $\mathbf{D}^{\mathbf{t_i}}(t) = \mathbf{s}(t) \cdot \mathbf{v}^{t_i}(t)$ (the vector of visit ratios $\mathbf{v}^{t_i}$ is normalized for transition t_i). In the following, we omit the superindex t_i in $\mathbf{D}^{\mathbf{t_i}}$ for clarity.

After some manipulations of (8), a lower bound for the *average interfiring time* of transition t_i, i.e., $\mathbf{\Gamma}^{\mathbf{lb}}(t_i)$, can be computed by solving the following LP problem (LPP) [27]:

$$\begin{aligned} \mathbf{\Gamma}(t_i) \geq \mathbf{\Gamma}^{\mathbf{lb}}(t_i) = \text{maximum } & \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D} \\ \text{subject to } & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & \mathbf{y} \geq \mathbf{0}. \end{aligned} \tag{9}$$

As a side product of the solution of (9), $\mathbf{y}$ represents the *slowest* p-semiflow of the system; thus, LPP (9) can also be seen as a search for the most constraining p-semiflow. This p-semiflow will be the one with the highest ratio $\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}/(\mathbf{y} \cdot \mathbf{m_0})$. Therefore, upper bound $\Theta(t_i)$ for the steady-state throughput can be calculated as the inverse of the lower bound for the average interfiring time $\mathbf{\Gamma}^{\mathbf{lb}}(t_i)$, i.e., $\Theta(t_i) = 1/(\mathbf{\Gamma}^{\mathbf{lb}}(t_i))$.

B. Next Slowest p-Semiflow

The LPP shown in (9) was the basis in [14] for developing an iterative algorithm to compute upper bounds in stochastic MGs. Unfortunately, the proposed algorithm is not applicable to more general nets than MGs, hence our search for an alternative method.

The new algorithm will follow a similar strategy. First, the initial bottleneck is computed using (9). Then, in each iteration step, the next slowest p-semiflow connected to the subnet associated to the current bottleneck is added to it.

Let us suppose that the p-semiflow $\mathbf{y}^*$ represents the initial bottleneck, i.e., $\mathbf{y}^*$ is obtained from the solution of (9). The following constraint imposes that some other p-semiflow $\mathbf{y}$, $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, is connected to $\mathbf{y}^*$: $\sum_{p \in V} \mathbf{y}(p) > 0$, where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$ (i.e., there exist places in the support of $\mathbf{y}$, which share output transitions with places in the support of $\mathbf{y}^*$). Hence, the p-semiflow $\mathbf{y}$ with the highest ratio $\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}/(\mathbf{y} \cdot \mathbf{m_0})$ connected to $\mathbf{y}^*$ can be searched for by solving the following LPP:

$$\begin{aligned} \text{maximum } & \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D} \\ \text{subject to } & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & \mathbf{y}(p) > 0 \qquad \forall p \in Q \\ & \sum_{p \in V} \mathbf{y}(p) > 0 \end{aligned} \tag{10}$$

where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$, and $Q = \{q \in P, q \in \|\mathbf{y}^*\|\}$.

As a result of LPP (10), we will obtain the p-semiflow $\mathbf{y}$, which will be a linear combination of $\mathbf{y}^*$ and the next most constraining p-semiflow.

The strict inequality in (10) could lead us to numerical problems since the lower the value of $\sum_{p \in V} \mathbf{y}(p)$, the higher the value of the optimization function. The Appendix discusses this issue and shows that the solution proposed in the following can be applied in practice. A way to solve this is by reformulating $\sum_{p \in V} \mathbf{y}(p) > 0$ into $\sum_{p \in V} \mathbf{y}(p) \geq h$, where h is strictly positive. The problem now is to set an appropriate value for h. A high value can make constraints $\mathbf{y} \cdot \mathbf{m_0} = 1$ and $\sum_{p \in V} \mathbf{y}(p) \geq h$ incompatible, leading to an infeasible LPP. A valid value of h can be obtained by searching for a real number that is lower than each component of a p-semiflow $\mathbf{y}$ that covers all places and satisfies $\mathbf{y} \cdot \mathbf{m_0} = 1$. Such a value can be obtained by means of the following LPP:

$$\begin{aligned} \text{maximum } & h \\ \text{subject to } & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & \mathbf{y} \geq h \cdot \mathbf{1} \\ & h > 0 \end{aligned} \tag{11}$$

where $\mathbf{1}$ is a vector with all entries equal to 1.

```
Input: $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, $\varepsilon$
Output: $\Theta$, $\mathbf{Q}$
 1: $\{\Theta, \mathbf{y}\}$ = Upper throughput bound and components of the
    initial bottleneck of $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$ according to (9)
 2: Calculate value $h$ by solving LPP (11)
 3: $\Theta' = 0$; $\mathbf{Q} = \{p \in P, p \in \|\mathbf{y}\|\}$
 4: while $\frac{\Theta - \Theta'}{\Theta} \geq \varepsilon$ and $\mathbf{Q} \neq P$ do
 5:     $\mathbf{V} = \{v | v \in {}^\bullet(\mathbf{Q}^\bullet) \setminus \mathbf{Q}\}$
 6:     $maximum\ \mathbf{y}' \cdot \mathbf{Pre} \cdot \mathbf{D}$
        $subject\ to\ \mathbf{y}' \cdot \mathbf{C} = \mathbf{0}$
            $\mathbf{y}' \cdot \mathbf{m_0} = 1$
            $\mathbf{y}'(p) \geq h,\ \forall p \in \mathbf{Q}$
            $\sum_{p \in V} \mathbf{y}'(p) \geq h$
 7:     $\Theta' = \Theta$
 8:     $\Theta$ = Throughput of the net composed by the p-semiflow
        $\mathbf{y}'$
 9:     $\mathbf{Q} = \{p \in P, p \in \|\mathbf{y}'\|\}$
10: end while
```

Fig. 2. Iterative strategy algorithm for computing upper throughput bounds.

The obtained value h ensures the feasibility of the following LPP, which is just a reformulation of (10):

$$\begin{aligned} &\text{maximum } \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D} \\ &\text{subject to } \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ &\qquad \mathbf{y} \cdot \mathbf{m_0} = 1 \\ &\qquad \mathbf{y}(p) \geq h \qquad \forall p \in Q \\ &\qquad \sum_{p \in V} \mathbf{y}(p) \geq h \end{aligned} \tag{12}$$

where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$, and $Q = \{q \in P, q \in \|\mathbf{y}^*\|\}$.

As has been said, the last constraint $\sum_{p \in V} \mathbf{y}(p) \geq h$ imposes that the support of $\mathbf{y}$ corresponds to the p-semiflow connected to $\mathbf{y}^*$ with the highest $\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}/(\mathbf{y} \cdot \mathbf{m_0})$.

C. Iterative Strategy to Compute Upper Throughput Bounds

This subsection presents an iterative strategy to obtain an improved upper throughput bound in TPPNs. In the first step, the strategy calculates the initial throughput bound of the system with the LPP (9) and takes the subnet associated to $\mathbf{y}$ as the initial bottleneck. In each iteration, the subnet associated to the p-semiflow that is potentially more constraining than the others is added to the bottleneck. The throughput is then calculated. Note that such an addition in each iteration restricts the behavior of the system, which implies a lower throughput. The iteration process stops when no significant improvement of the bound is achieved.

The algorithm in Fig. 2 represents the strategy used to compute throughput upper bounds. For the input, the algorithm needs the TPPN system to be analyzed, i.e., $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, and a degree of precision ($\varepsilon > 0$) to be achieved. As output, the upper throughput bound Θ and the places belonging to the bottleneck of the TPPN $\mathbf{Q}$ are obtained. The degree of precision ε will be used for the stopping criterion of the iterative strategy.

In the first place, the initial upper throughput bound is calculated by LPP (9) (step 1). Then, the value of h is computed by using the LPP shown in (11) so that the feasibility of the LP is ensured. The iteration process (steps 4–9) is repeated until no significant improvement is achieved with respect to the last iteration or until the last obtained bottleneck contains all places in its support. In the worst case, only one place will be added in each iteration. Therefore, the algorithm complexity is polynomial due to the LP.

In step 5, the places that share output transitions with some place contained in the support of $\mathbf{y}$ are calculated. Step 6 corresponds to the LPP (12). Finally, in step 8, the throughput of the subnet associated to the new bottleneck is considered as the new upper bound. The throughput is calculated by solving the Markov chain [25] associated to the current bottleneck when it can be computed within a practical time, or by simulation otherwise.

Example: Consider again the supermarket example shown in Fig. 1. Let the initial marking be $nC = 21$, $nS = 4$, and $nP = 2$. The vector of visit ratios $\mathbf{v}$ normalized for transition t_1 is $\mathbf{v}^{t_1} = \{1.0, 1.0, 1.0, 0.4, 0.6, 0.6, 0.6, 0.6, 1.0, 1.0\}$. According to LPP (9) (step 1 of the algorithm in Fig. 2), the critical bottleneck is composed of $\|\mathbf{y}\| = \{p_0, p_1, p_3, p_4, p_5, p_7, p_8, p_9, p_{10}\}$, i.e., the p-semiflow, which corresponds to the customers' life cycle. Such a result indicates that the system has, on average, enough resources to attend to the expected incoming customers. The upper throughput bound (normalized for transition t_1) of the critical bottleneck is $\Theta(t_1) = 0.567521$ [result of LPP (9)], and the value that guarantees the feasibility of the problem is $h = 0.037037$ (step 2). The places sharing output transitions with places in $\|\mathbf{y}\|$, i.e., connected to the critical bottleneck, are p_2 and p_6 (calculated in step 5). Each one corresponds to the resources of the system, the supermarket cashiers, and PoS terminals, respectively. The result of the LPP in step 6 allows to regrow the current bottleneck, imposing that $\mathbf{y}'(p_2) + \mathbf{y}'(p_6) \geq h$ (i.e., one of them, at least, must be contained on the support of $\mathbf{y}'$), and gives the new bottleneck that is composed of $\|\mathbf{y}'\| = \{p_0, p_1, p_2, p_3, p_4, p_5, p_7, p_8, p_9, p_{10}\}$. The new throughput is $\Theta'(t_1) = 0.514220$ (step 8), which represents an improvement of 9.3919% with respect to the previous bottleneck. Note that the place added is that representing the number of cashiers (i.e., p_2).

Let us assume that $\varepsilon = 0.001$. As the relative difference between Θ and Θ' is 0.093919 (as commented previously), the iteration process carries on. At this point, the only place that is not connected to the critical bottleneck is p_6, which corresponds to the number of PoS terminals. By solving the LPP in step 6, the new bottleneck is obtained, which has all the places of the system in its support (i.e., $\|\mathbf{y}\| = P$), and the new throughput is $\Theta = 0.480642$. Therefore, as the support of the new bottleneck contains all places of the net, the iteration process finishes. The new throughput Θ represents an improvement of 6.5299% with respect to the previous bottleneck and a total improvement of 15.3085% with respect to the initial bottleneck.

The proposed iterative strategy is applied to a larger system in Section VI.

V. Resource Optimization

Here, we propose a heuristic strategy to gauge the number of resources that a system should allocate. Our approach for resource optimization is similar to Goldratt's principle [29]: Once the system's bottleneck is identified, the associated resource is increased.

A. Calculating the Next Constraining Resource

Let us recall LPP (9) to calculate an upper throughput bound. The most constraining p-semiflow $\mathbf{y}$ will have just one marked place in its support due to the net structure (as explained in Section III). Assume that the marked place corresponds to a resource place (not the process-idle place). Then, given that $\mathbf{y}$ constrains the throughput of the whole system, the addition of more instances to the resource place will result in an increase in the system throughput. At a certain moment, the resource becomes saturated, and adding more instances does not improve the throughput. This occurs because the constraining p-semiflow has changed. Note that the upper throughput bound will linearly increase with the number of tokens of the resource place because it is the only place in $\|\mathbf{y}\|$ having tokens and the equation $\mathbf{y}\cdot\mathbf{Pre}\cdot\mathbf{D}$ is linear.

Hence, resource r_1 contained in the support of the most constraining p-semiflow $\mathbf{y}_{r_1}$ can be increased until $\mathbf{y}_{r_1}$ is no longer the bottleneck p-semiflow. Let $\mathbf{m}_\mathbf{0}^\triangle$ be the initial marking vector $\mathbf{m_0}$ with an increase α_1 of the resource r_1, i.e.,

$$\mathbf{m}_\mathbf{0}^\triangle = \begin{cases} \mathbf{m_0}(p), & p \neq r_1 \\ \mathbf{m_0}(p)+\alpha_1, & p = r_1. \end{cases} \tag{13}$$

The p-semiflow $\mathbf{y}_{r_1}$ is not the only constraining p-semiflow if the following equation holds:

$$\frac{\mathbf{y}_{r_1}\cdot\mathbf{Pre}\cdot\mathbf{D}}{\mathbf{y}_{r_1}\cdot\mathbf{m}_\mathbf{0}^\triangle} \leq \frac{\mathbf{y}_{r_2}\cdot\mathbf{Pre}\cdot\mathbf{D}}{\mathbf{y}_{r_2}\cdot\mathbf{m}_\mathbf{0}^\triangle} \tag{14}$$

where $\mathbf{y}_{r_2} \neq \mathbf{y}_{r_1}$ is a p-semiflow. Note that the p-semiflow $\mathbf{y}_{r_2}$ will contain in its support the next most constraining resource r_2 and, by definition, $r_1 \neq r_2$.

The number α_1 of instances of the resource place r_1, contained in the most constraining p-semiflow $\mathbf{y}_{r_1}$, which need to be added to obtain the next constraining resource r_2, contained in the next most constraining p-semiflow $\mathbf{y}_{r_2}$, can be easily computed by solving the following LPP:

$$\begin{aligned} \text{minimum} \quad & \alpha_1 \\ \text{subject to} \quad & \mathbf{y}_{r_2}\cdot\mathbf{Pre}\cdot\mathbf{D} = \mathbf{y}_{r_1}\cdot\mathbf{Pre}\cdot\mathbf{D} \\ & \mathbf{y}_{r_2}\cdot\mathbf{C} = 0 \\ & \mathbf{y}_{r_2}(r_1) = 0 \\ & \mathbf{y}_{r_2}\cdot\mathbf{m}_\mathbf{0}^\triangle = \mathbf{y}_{r_1}\cdot\mathbf{m}_\mathbf{0}^\triangle \\ & \mathbf{m}_\mathbf{0}^\triangle = \begin{cases} \mathbf{m_0}(p), & p \neq r_1 \\ \mathbf{m_0}(p)+\alpha_1, & p = r_1 \end{cases} \\ & \alpha_1, \mathbf{y}_{r_2} \geq 0 \end{aligned} \tag{15}$$

where $\mathbf{y}_{r_1}$ is the p-semiflow, which contains r_1 in its support; $\mathbf{y}_{r_2}$ is the p-semiflow, which contains r_2 in its support; and $\mathbf{m}_\mathbf{0}^\triangle$ represents the initial marking vector $\mathbf{m_0}$ with the increase α_1 in r_1.

Constraints $\mathbf{y}_{r_2}\cdot\mathbf{Pre}\cdot\mathbf{D} = \mathbf{y}_{r_1}\cdot\mathbf{Pre}\cdot\mathbf{D}$ and $\mathbf{y}_{r_2}\cdot\mathbf{m}_\mathbf{0}^\triangle = \mathbf{y}_{r_1}\cdot\mathbf{m}_\mathbf{0}^\triangle$ are both parts (dividend and divisor, respectively) of (14) equalled. Constraint $\mathbf{y}_{r_2}\cdot\mathbf{C} = 0$ ensures that $\mathbf{y}_{r_2}$ is a left anuler of the incidence matrix, hence a p-semiflow of the net. Finally, constraint $\mathbf{y}_{r_2}(r_1) = 0$ is added to avoid a product of two optimization variables (the variable α_1 and the variable $\mathbf{y}_{r_2}(r_1)$ in equation $\mathbf{y}_{r_2}\cdot\mathbf{m}_\mathbf{0}^\triangle = \mathbf{y}_{r_1}\cdot\mathbf{m}_\mathbf{0}^\triangle$). Moreover, variable $\alpha_1 \in \mathbb{R}_{\geq 0}$; therefore, the linearity of the optimization problem is ensured.

Both α_1 and the next constraining p-semiflow $\mathbf{y}_{r_2}$ are obtained when the LPP is solved. Note that the increase in resource r_1 does not affect the ratio $\mathbf{y}\cdot\mathbf{Pre}\cdot\mathbf{D}/(\mathbf{y}\cdot\mathbf{m_0})$ of any other minimal p-semiflow $\mathbf{y}$, which contains another resource in its support (see definition of the process PNs class in Section III). Notice that, as in Section IV, an LPP is used to solve a problem that deals with integer values as the number of resources. This relaxation of the real domain remarkably decreases the complexity of the approach (the complexity of solving a LPP is polynomial), at the cost of some loss of precision in the results. Once both α_1 and the next constraining p-semiflow $\mathbf{y}_{r_2}$ are obtained, LPP (15) can be easily extended to calculate the next constraining resource and the number of tokens, i.e., instances, to be increased of both places as follows:

$$\begin{aligned} \text{minimum} \quad & \alpha_1 + \alpha_2 \\ \text{subject to} \quad & \mathbf{y}'\cdot\mathbf{Pre}\cdot\mathbf{D} = \mathbf{y}_{r_1}\cdot\mathbf{Pre}\cdot\mathbf{D} \\ & \mathbf{y}'\cdot\mathbf{C} = 0 \\ & \mathbf{y}'(r_1) = 0 \\ & \mathbf{y}'(r_2) = 0 \\ & \mathbf{y}'\cdot\mathbf{m}_\mathbf{0}^\triangle = \mathbf{y}_{r_1}\cdot\mathbf{m}_\mathbf{0}^\triangle \\ & \mathbf{y}'\cdot\mathbf{m}_\mathbf{0}^\triangle = \mathbf{y}_{r_2}\cdot\mathbf{m}_\mathbf{0}^\triangle \\ & \mathbf{m}_\mathbf{0}^\triangle = \begin{cases} \mathbf{m_0}(p), & p \notin \{r_1, r_2\} \\ \mathbf{m_0}(p)+\alpha_1, & p = r_1 \\ \mathbf{m_0}(p)+\alpha_2, & p = r_2 \end{cases} \\ & \alpha_1, \alpha_2, \mathbf{y}' \geq 0 \end{aligned} \tag{16}$$

where $\mathbf{m}_\mathbf{0}^\triangle$ represents the initial marking vector $\mathbf{m_0}$ with the increase α_1 of place r_1 and the increase α_2 of place r_2, and $\mathbf{y}_{r_1}$ ($\mathbf{y}_{r_2}$) is the p-semiflow, which contains r_1 (r_2) in its support.

As in LPP (15), constraint $\mathbf{y}'\cdot\mathbf{C} = 0$ ensures that $\mathbf{y}'$ is a left annuler of the incidence matrix; hence, $\mathbf{y}'$ is a p-semiflow of the net. In addition, constraints $\mathbf{y}'(r_1) = 0$ and $\mathbf{y}'(r_2) = 0$ ensure linearity of the optimization problem. Constraints $\mathbf{y}'\cdot\mathbf{m}_\mathbf{0}^\triangle = \mathbf{y}_{r_1}\cdot\mathbf{m}_\mathbf{0}^\triangle$ and $\mathbf{y}'\cdot\mathbf{m}_\mathbf{0}^\triangle = \mathbf{y}_{r_2}\cdot\mathbf{m}_\mathbf{0}^\triangle$ are the key of this LPP because both values of α_1 and α_2 can be obtained from these equations.

Note that $\mathbf{y}'\cdot\mathbf{Pre}\cdot\mathbf{D} = \mathbf{y}_{r_2}\cdot\mathbf{Pre}\cdot\mathbf{D}$ is not a constraint in LPP (16). This is a consequence of the result of LPP (15): From the latter LPP where r_1 is calculated, it is imposed that $\mathbf{y}_{r_2}\cdot\mathbf{Pre}\cdot\mathbf{D} = \mathbf{y}_{r_1}\cdot\mathbf{Pre}\cdot\mathbf{D}$. The addition of this constraint does not add new information to LPP (16).

LPP (16) can be generalized for more resources, as is shown in step 5 of the algorithm in Fig. 3.

B. Iterative Strategy for Resource Optimization

This subsection presents a heuristic iterative strategy that aims at maximizing the throughput by increasing the number of

```
Input: $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, $R$, $p_0$, $budget$, $c$
Output: $n$
1: Calculate initial bottleneck $\mathbf{y}_1$ by solving LPP (9)
2: $k = 0$; $cost = 0$; $n' = \mathbf{0}$
3: while $cost < budget$ and $k \neq |R|$ and $\|\mathbf{y_{k+1}}\| \cap \{p_0\} = \emptyset$ do
4:     $k = k + 1$; $cost' = cost$; $n = n'$; $\mathbf{A} = \{p | p \in P, p \in \|\mathbf{y}_j \cap R\|\}$, $\forall j \in \{1 \ldots k\}$
5:     $minimum \sum_{j=1}^{k} \alpha_j$
       $subject\ to\ \mathbf{y_{k+1}} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y_1} \cdot \mathbf{Pre} \cdot \mathbf{D}$
           $\mathbf{y_{k+1}} \cdot \mathbf{C} = \mathbf{0}$
           $\mathbf{y_{k+1}} \cdot \mathbf{m_0^{\Delta}} = \mathbf{y_j} \cdot \mathbf{m_0^{\Delta}}, \ \forall j \in \{1 \ldots k\}$
           $\mathbf{m_0}^{\Delta} = \begin{cases} \mathbf{m_0}(p) + \alpha_j, & p \in \mathbf{A} \\ \mathbf{m_0}(p), & otherwise \end{cases}$
           $\mathbf{y_{k+1}}(p) = 0, \ p \in \mathbf{A}$
           $\mathbf{y_{k+1}}, \ \alpha_j \geq 0, \ \forall j \in \{1 \ldots k\}$
6:     $cost = 0$; $n' = \mathbf{0}$
7:     for each $\alpha_j$, $\forall j \in \{1 \ldots k\}$ do
8:         $r_j = \|\mathbf{y}_j\| \cap R$; $n'_j = \lceil \alpha_j \rceil$
9:         $cost = cost + \lceil \alpha_j \rceil \cdot c_i$
10:    end for each
11: end while
12: if $k \leq |R|$ and $cost \leq budget$ then
13:     $n = n'$
14: end if
15: if $k < |R|$ and $cost \leq budget$ and $\|\mathbf{y_{k+1}}\| \cap \{p_0\} = \emptyset$ then
16:     $assignRestOfBudget(budget - cost, \langle \mathcal{S}, \mathbf{s} \rangle, R, c, n)$
17: end if
```

Fig. 3. Resource optimization heuristic strategy algorithm.

resources appropriately. The main idea of the strategy is to estimate the *inflexion points* where the constraining p-semiflows change and, hence, to estimate the increase in resources needed. More precisely, each unit of a resource has an associated cost, and the strategy establishes how to spend a given budget such that the throughput is maximized. The strategy ends either when there is no budget to spend, all resources have been dimensioned, or the last computed p-semiflow indicates an increase in the process-idle place.

The algorithm in Fig. 3 shows the resource optimization heuristic strategy. For the input, the algorithm needs the TPPN system to be analyzed, i.e., $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$. The sets of resources and the process-idle place of the system are denoted by R and p_0, respectively. The assigned budget to be spent is denoted by $budget$. The vector of cost is denoted by c, which assigns a cost c_i to each resource r_i contained in R. The output is the number of items n_i needed to increase each resource r_i.

First, an upper throughput bound $\mathbf{y}_1$ of $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$ is calculated according to LPP (9). After that, the iteration process (steps 3–10) is repeated either until the last assignment of resources has spent the available budget, until all resources have been dimensioned, or until the last computed resource to be increased matches with the process-idle place.

Step 5 calculates, in each iteration, the number of items of a resource that need to be increased to obtain the next restrictive resource. It should be noted that the LPP in step 5 is a generalization of LPP (15). After that, the cost of increasing such a number of instances of the resources is computed. Note that the ceiling integer of the value α_j is taken as the result. There are two reasons for this: First, we assume that the number of instances of the resources must be a natural number; and second, when the resource is not saturated, it will still be the restrictive resource.

Finally, step 12 checks whether all the resources have been assigned and that the cost of new resources does not exceed the given budget. When these conditions are fulfilled, the last resource assignment is taken as the valid one. Step 15 checks whether there is any resource that has not been assigned, the last resource assignment does not exceed the given budget, and the last computed p-semiflow does not contain the process-idle place. When these conditions are fulfilled, the remaining budget may be spent on increasing the system throughput. A procedure is invoked (assignRestOfBudget, step 16) for spending the rest of the assigned budget to increase the resources as much as possible. Note that the assignment of the remaining budget is an NP-problem, similar to the bounded knapsack problem [30]. To solve it, several heuristics can be used. For instance, a "round-trip" algorithm, which tries to increase all the resources per round until it cannot longer increase them.

Let us illustrate the use of this strategy through the supermarket example, as depicted in Fig. 1. Suppose that an initial marking of $nC = 30$, $nS = 2$, and $nP = 2$, and an initial budget of \$30 000. Hiring a new supermarket cashier costs \$5000, whereas a new PoS terminal has a price of \$700. The initial bottleneck is $\|\mathbf{y_1}\| \cap R = \{p_{nS}\}$, i.e., the subnet associated to the customers' cashiers. Therefore, this result gives us the following information: To attend to 30 customers whose think time follows an exponential distribution of a mean of 30 min, more supermarket cashiers are needed. The LPP at step 5 gives, in the first iteration, the increase in new cashiers needed, i.e., $\alpha_1 = 2.666$, and the new constraining p-semiflow, which corresponds to the use of the PoS terminals. Therefore, at least three new cashiers $(\lceil \alpha_1 \rceil)$ are needed to attend to the customers.

As the cost of hiring new cashiers is \$5000 and the initial budget is \$30 000, the new hirings can be done, and there is still money that remains to be spent; therefore, a new iteration can take place. The LPP at step 5 gives, in the second iteration, the values of $\alpha_1 = 3.6752$ and $\alpha_2 = 0.4322$. Hence, to attend to the customers, four new hirings and one more PoS terminal are needed. As the cost of these are \$20 700 in total, the increase in resources can be carried out. Now, the unassigned budget is \$9300, and we can continue increasing both resources in parallel. Indeed, the relation between both resources is known due to the equalities of the ratios.

In this case, although part of the budget remains to be spent, the new constraining p-semiflow contains the process-idle place, i.e., the place representing customers. Thus, the resources of the system (cashiers and PoS terminals) have been optimally calculated to attend to 30 customers whose think time follows an exponential distribution of a mean of 30 min. This way, the algorithm has calculated that, to attend to the customers, at least four more supermarket cashiers and one PoS terminal are needed.

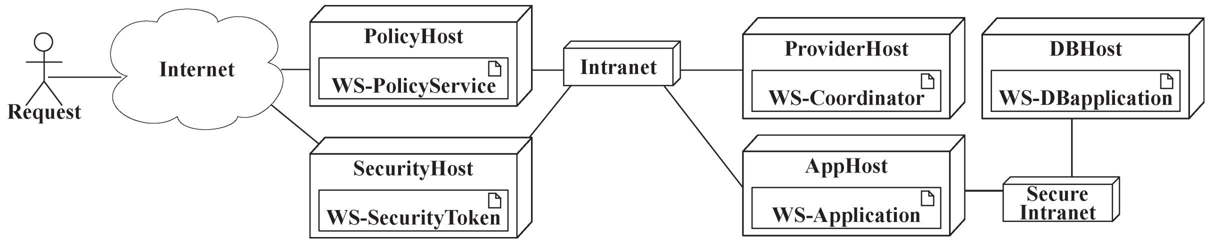


Fig. 4. SDBS deployment.

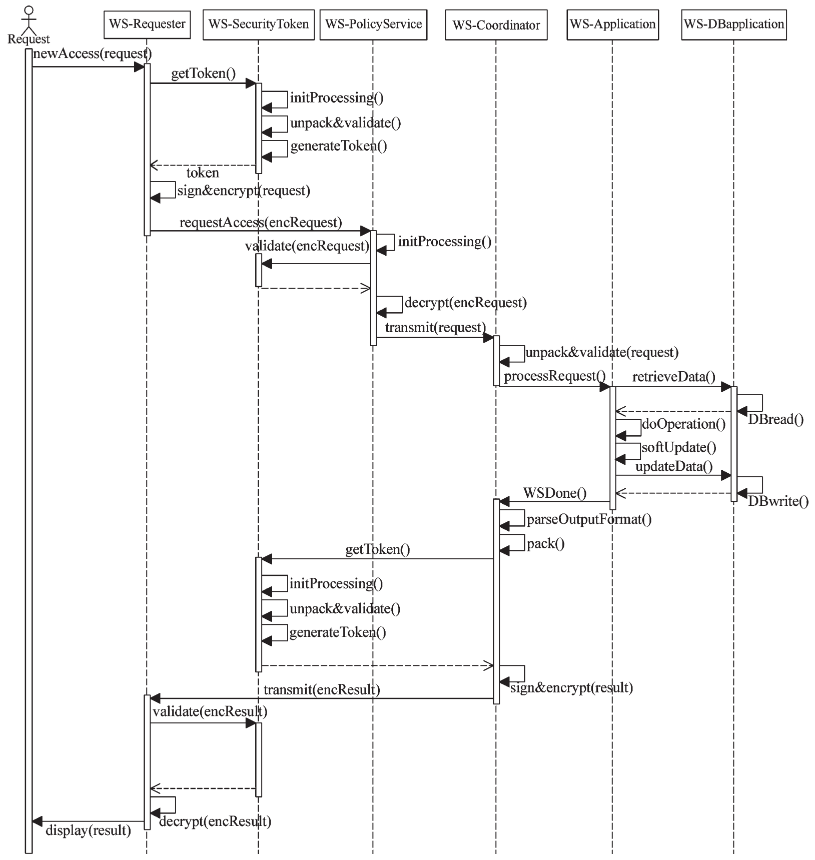


Fig. 5. SDBS update customer data scenario.

Note that it may happen that the LPP at step 5 returns the p-semiflow containing in its support the process-idle place in the first iteration. This would indicate that the system has enough resources to attend to such a number of customers with such a think time. Therefore, the strategy is also able to compute when a system with an initial configuration is able to support the estimated workload or, otherwise, to compute the number of instances of resources needed to be able to support such as a workload.

VI. Case Study: A SDBS

In this section, we introduce a case study to test our approach. We consider the design of a secure database system (SDBS) deployed as a web service, which stores sensitive information. There are users who will eventually have access to this information. A real application of this kind of system is, for instance, a web server storing customer data of an insurance company or a bank web server keeping customers' account balances.

A. Description

Fig. 4 shows the actual deployment of the SDBS, which includes the hardware resources (depicted as cubes) and their network links (arrows between cubes or cubes in the case of intranets). Software modules (depicted as squares) are deployed within in hardware resources. The architecture of the system is as follows: There is a policy host, a security host, a provider host, an application host, and a DB host. Moreover, the latter is isolated and reachable only through a secure intranet connected to the application host. Note that each of these hosts deploys a specific service or a software module.

Following Fig. 5, the SDBS works as follows: A user interacts with an application outside the system ($\texttt{WS}-\texttt{Requester}$), which collects his/her personal data and the type of operation required (let us assume it is an update of personal user data) by the user. This information is summarized in a request. Each request coming into the system needs a security token to be identified before accessing the system, provided by the security host through the $\texttt{WS}-\texttt{SecurityToken}$ service. Once

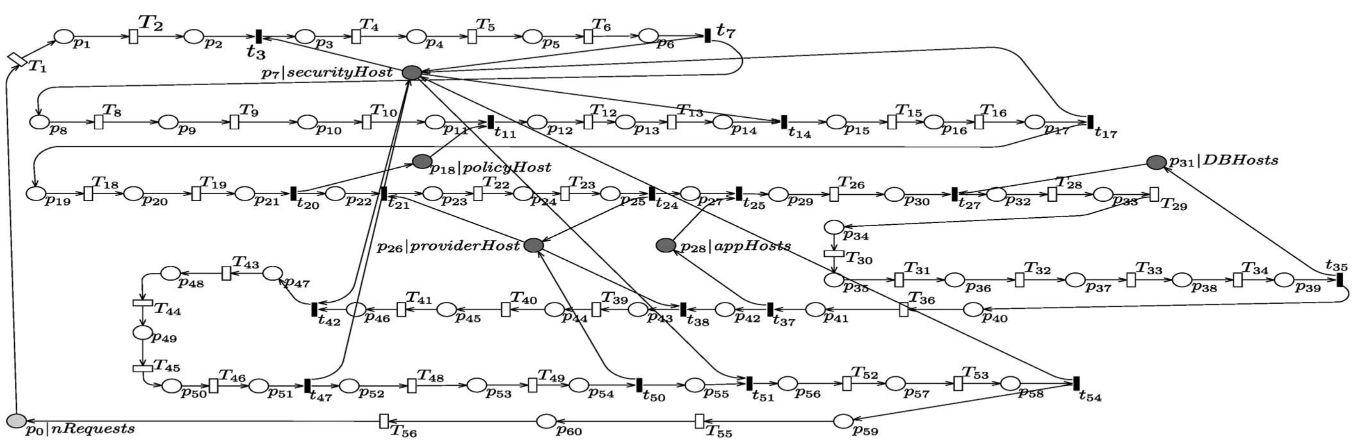


Fig. 6. Petri net of the SDBS. Resource places are depicted in dark gray and process-idle place in light gray.

TABLE I
EXPERIMENTAL PARAMETERS

Transition	Parameter	Value (s)
T_1	*$reqRate*	$0.2ms$
T_2, T_8, T_{10}, T_{49}, T_{53}	*$delayNet*	$2.5ms$
T_{13}, T_{16}, T_{19}, T_{23}, T_{36}, T_{41}, T_{46}	*$intranetLag*	$0.2ms$
T_{26}, T_{29}, T_{31}, T_{34}	*$secIntraLag*	$0.5ms$
T_4, T_{43}	*$initProc1*	$1ms$
T_{15}, T_{22}, T_{52}	*$validate*	$0.3ms$
T_6, T_{45}	*$genToken*	$0.5ms$
T_9, T_{48}	*$sign*	$0.8ms$
T_{18}, T_{55}	*$decrypt*	$1ms$
T_{12}	*$initProc2*	$0.3ms$
T_5, T_{44}	*$unpack*	$0.1ms$
T_{40}	*$pack*	$0.1ms$
T_{39}	*$parseOutput*	$0.3ms$

(a)

Place	Parameter	Value (s)
p_0	*$nRequests*	$\{15, 20, 21, 22, 23 \dots 30\}$
p_7	*$nSec*	5
p_{18}	*$nPolicy*	10
p_{26}	*$nCoords*	10
p_{28}	*$nApps*	5
p_{31}	*$nDBapps*	2

(b)

Trans.	Parameter	Value (s)
T_{28}	*$DBread*	$0.2ms$
T_{30}	*$perform*	$0.6ms$
T_{32}	*$consistency*	$0.2ms$
T_{33}	*$DBupdate*	$0.2ms$
T_{56}	*$processResult*	$1.5ms$

(c)

the security token is retrieved, the request is accordingly signed. Access is then requested from the policy host that checks the request ($\texttt{WS} - \texttt{PolicyService}$). When permission is granted, the $\texttt{WS} - \texttt{Coordinator}$ service is invoked. This communicates with the $\texttt{WS} - \texttt{Application}$ service (located in the application host). The latter host has access to the DB application ($\texttt{WS} - \texttt{DBapplication}$) through a secure intranet. Finally, the DB application definitively updates the user request into the DB, and an acknowledgement report is returned back through the system.

The PN representing the behavior of the SDBS system is depicted in Fig. 6. Each resource is represented by a dark gray place in the PN, i.e., p_7 (policy host), p_{18} (security host), p_{26} (provider host), p_{28} (application host) and p_{31} (DB host). whereas user's requests are represented by the process-idle place p_0 (depicted in light gray). The number of instances of each resource is summarized in Table I(b), and they will be represented by tokens in the respective place. Note that different numbers of tokens in p_0 will be used for sensitive analysis in experiments.

The acquisition (release) of a resource is represented by an immediate transition with an input (output) arc. For instance, transition t_3 represents the acquisition of the security host, whereas t_7 represents the release of such a resource.

Each one of the activities, shown as self-messages in Fig. 5, has been transformed into an exponential transition in the PN with its corresponding duration [given in Table I(a) and (c)]. Each message exchanged through a net between two resources [e.g., getToken()] gives rise in the PN to an exponential transition (e.g., T_2) whose delay is that of the net involved (e.g., $delayNet). We have assumed that the operations/messages needed for establishing communication through the secure intranet are more expensive (in computing time terms). For this reason, we have set a higher delay for the secure intranet ($intranetLag) than for the insecure intranet ($secIntraLag). For the sake of simplicity, we have assumed the same delay for each message on the intranet communications, irrespective of its size.

The workload is defined by the number of requests from users concurrently accessing the SDBS, which is parameterized by the variable $nRequests, which is an input parameter for the analysis. The number of hosts (security host, policy host, etc.) has been indicated using variables ($nSec, $nPolicy, etc.). Finally, the throughput of the intranets is considered through variables $intranetLag and $secIntraLag. The values for all these input variables used for the experiments described in the following and their corresponding transitions/places on the PN appear in Table I.

B. Experiments and Discussion

Here, we test our approach by performing a set of experiments in the PN that accurately represents the SDBS. After applying our approach, the results obtained will be discussed.

1) Performance Estimation: We have carried out the regrowing strategy (algorithm in Fig. 2, Section IV-C) to estimate the throughput of the SDBS system with a different number of requests. The overall strategy has been implemented in Matlab, whereas the throughput computation of the SDBS has been

TABLE II
EXPERIMENTAL RESULTS FOR NUMBER OF REQUESTS $\{15, 20, 21, 22, 23, \ldots, 30\}$

Number of requests	**Regrowing step**	**Size** $\lvert P\rvert$ (%)	$\lvert T\rvert$ (%)	**Through-put**	**Partial improvement**	**Bound error**	**Execution time (s)**
15	(full system)	61 (100%)	56 (100%)	0.525685			$> +1$ day
	(initial bound)	56 (91.80%)	56 (100%)	0.551637	-	4.7045%	$5.87s$
	1	57 (93.44%)	56 (100%)	0.533037	3.3718%	1.3792%	$122.94s$
	2	58 (95.08%)	56 (100%)	0.522379	1.9995%	−0.6330%	$751.20s$
	3	59 (96.72%)	56 (100%)	0.522346	0.0063%	−0.6393%	$34256.97s$
20	(full system)	61 (100%)	56 (100%)	0.652313			$> +1$ day
	(initial bound)	56 (91.80%)	56 (100%)	0.735930	-	11.3621%	$5.80s$
	1	57 (93.44%)	56 (100%)	0.675957	8.1493%	3.4979%	$302.60s$
	2	58 (95.08%)	56 (100%)	0.637812	5.6431%	−2.2735%	$300.17s$
	3	59 (96.72%)	56 (100%)	0.637860	−0.0075%	−2.2658%	$3166.09s$
21	(full system)	61 (100%)	56 (100%)	0.671806			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	9.3063%	$0.18s^\dagger$
	1	57 (93.44%)	56 (100%)	0.697133	5.8871%	3.6331%	$826.82s$
	2	58 (95.08%)	56 (100%)	0.653556	6.2509%	−2.7924%	$280.46s$
	3	59 (96.72%)	56 (100%)	0.653116	0.0673%	−2.8616%	$2216.06s$
22	(full system)	61 (100%)	56 (100%)	0.687808			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	7.1459%	$0.18s^\dagger$
	1	57 (93.44%)	56 (100%)	0.713762	3.6422%	3.6362%	$2763.5s$
	2	58 (95.08%)	56 (100%)	0.666148	6.6709%	−3.2515%	$502.95s$
	3	59 (96.72%)	56 (100%)	0.667222	−0.1612%	−3.0853%	$1502.62s$
$23 \ldots 30$	(full system)	61 (100%)	56 (100%)	0.700056			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	5.4925%	$0.18s^\dagger$
	1	14 (22.95%)	13 (23.21%)	0.740733	0.0011%	5.4915%	$0.262s^\dagger$

performed with the GreatSPN tool. The GreatSPN tool has been run in an Intel Pentium IV 3.6 GHz with 2-GiB RAM DDR2 533 MHz host machine.

Table II shows the results obtained in the set of experiments with the parameters set as described earlier. The first column indicates the number of requests, followed by the number of *regrowing steps*. We have applied the name regrowing step to each iteration of the loop of the algorithm in Fig. 2. For each number of requests considered in the experiments, we have simulated the whole system. Such results are indicated in the first row of each experiment. The next column shows the size of the bottleneck (in terms of the number of places and transitions) produced by the algorithm and its percentage with respect to the total size. Then, the result of the upper throughput bound computed by the algorithm is shown. Such a bound is computed by solving the underlying Markov chain when this is computationally feasible [12] or by simulating the net otherwise. Note that, in the case of simulation, the upper throughput bound value is the mean of the simulation values, and the real upper throughput bound value is within an interval of ±4%, with a confidence level of 95%. The next two columns show, in the first place, the percentage of increasing/decreasing improvement of one bound with respect to the previous upper throughput bound and, second, the accuracy of the computed bound with respect to the throughput of the whole system. The negative relative errors are caused by the confidence level and degree of accuracy used in the experiments. Finally, the last column shows the execution time consumed for computing the upper throughput bound of the PN system. We have distinguished whether the computation of the upper throughput bound has been achieved by exact analysis († symbol) or by simulation (no symbol).

Note that, in all cases, the computation of the throughput of the whole system takes longer than one day of simulation time to finish, although the evaluated system is an academic example. For larger systems, simulations may need a long convergence time; therefore, the computation of the usefulness of bounds is proven.

The degree of precision ε of the algorithm in Fig. 2 has been set to 10^{-3}. As can be observed, the initial bottleneck with the lowest number of requests (15, 20) corresponds to the underlying state machine (this is the result of removing resource places from the net in Fig. 6). Again, this result indicates that the system's resources are well dimensioned for attending to such a number of requests. In the case of 15 requests, in each iteration step, there is no significant improvement (near to 6% in two iterations), and the regrowing strategy finishes in few steps. However, the greatest improvement occurs when the requests reach 20 units. In such a case, the first regrowing achieves an improvement near to 8%, reaching over 13% in the next iteration.

It is interesting to note what happens when the requests are increased to 21. For this value, the initial bottleneck is produced by one of the system's resources (specifically, the number of DB application hosts). This implies that the throughput bound of the system will remain the same for any number of requests over 21 (see *Average throughput* of the first regrowing step for a number of requests greater than 21). In other words, requests will start waiting to be attended to if their number is equal to or higher than 21. In addition, note that when the number of requests is greater than 23, in the second iteration step, there is an improvement in the upper throughput bound lower than 10^{-3}%.

As stated earlier, the most significant improvement occurs when the number of requests is 20. In just one iteration step, the initial throughput bound is improved by a value of nearly 8%. This indicates that the proposed method is more useful (i.e., it achieves a significant improvement in the upper throughput bound in few iterations) if the resources and requests are more well balanced. In addition, it should be noted that the simulation of the whole PN becomes unfeasible for large systems, as indicated by the execution time.

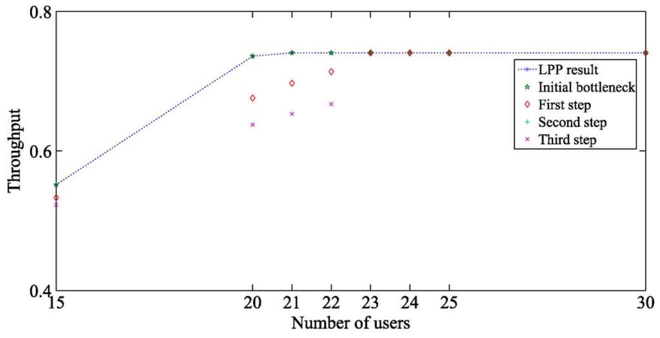


Fig. 7. Throughput of the SDBS with variable number of users.

The throughput results have been plotted in Fig. 7. The throughput is drawn for each number of requests and for each step. In addition, the result of LPP (9) has also been drawn (dotted line). The LPP values match the throughput values of the initial bottleneck. As expected, the result of solving the LPP (9) (dotted line) is an upper bound of all the rest of the values. As can be seen, the improvement in the upper throughput bound for each regrowing step is almost insignificant in the case of requests lower than 20 or greater than 25. While the number of requests is near to 20, the relative difference between the throughput of the initial upper throughput bound and the first iteration becomes greater, reaching its maximum in the case of 20 requests. After that point, it becomes lower even tending toward a minimal difference near to zero (see, for instance, the case of 30 requests).

Finally, the execution time shown in the last column in Table II indicates that the bigger the size of the net, the longer it takes to complete the simulation. Note that small additions to the net (i.e., just one place) normally cause an execution time of one or two orders of magnitude greater than the previous executions. However, the improvement of the upper throughput bound is not so significant as to justify such an amount of execution time.

The main conclusions that can be extracted from both experiments can be summarized as follows.

- There exists a number of requests (*inflexion point*) from which the initially most restrictive p-semiflow of the system changes. Around such an inflexion point, the accuracy of the initial throughput bound is low. This occurs because when the slowest p-semiflow of the system is much slower than the others, it predominates over them, and the system throughput is determined by the throughput of such a p-semiflow. The initial throughput bound is therefore usually quite accurate. However, when several p-semiflows have similar speeds, none of them predominates over the others. Hence, the initial throughput bound, which considers just one p-semiflow, is less accurate.
- The improvement in the upper bound is particularly significant in the proximity of the inflexion point.

As future work, we aim to continue researching into performance estimation based on performance bounds, seeking to obtain some quality bound characterization. The use of LP problems and the token/delay ratio between p-semiflows in a PN system could be useful for this goal.

As the reader can imagine, it would be of great interest to be able to compute such inflexion points directly. This is the goal in the next set of experiments.

2) Resource Optimization: For these experiments, the number of requests has been set to $nRequests = 100$, whereas the initial number of resources remains unchanged: five security hosts, ten policy hosts, ten coordination hosts, five application hosts, and two DB application hosts (summarized in Table I). Let us suppose a budget of \$20 000 and the following costs per resource: \$3500 per security host (represented by place p_7), \$1000 per policy host (place p_{18}), \$2000 per coordinator host (place p_{26}), \$500 per application host (place p_{28}), and \$500 per DB application host (place p_{31}). The prices of the hosts reflect either the cost of the physical hardware or the cost of reimplementing the services.

Applying the optimization strategy introduced in Section V, the initial restrictive resource is the number of DB application hosts, i.e., \$nDBapps (initial tokens of place p_{31}). The algorithm in Fig. 3 computes the new restrictive resource, the security hosts, and the number of DB application hosts needed to be increased (which is just one host). As the cost is \$500 per DB application host and there is a budget of \$20 000, the increase is possible. The strategy continues looking for the next restrictive resource. The second iteration gives as a result the new restrictive resource (application host) and the new instances of DB application and security hosts are two and five units, respectively. The increase in such resources has a cost of \$18 500; therefore, it can be afforded. The new restrictive resource after the third iteration is the number of coordinator hosts. This time, it is necessary to increase the security hosts by six units, the DB application hosts by three units, and the application hosts by one unit with respect to the initial configuration. This last assignment has a cost greater than the initial budget; therefore, the iteration process finishes, and the previous assignment is taken as the valid one (five security hosts and two DB application hosts). Moreover, there is no possibility of spending the rest of the budget (which amounts to \$1500). Therefore, the optimization strategy ends.

Hence, with the initial configuration and the given budget, the number of security hosts needs to be increased by five units and the number of DB application hosts by two units in order for the system resources to be optimally distributed and the throughput maximized.

Fig. 8 plots the upper throughput bound (dashed line) of each configuration of resources, its associated cost in dollars (dotted line), and the total assigned budget (solid line). The *initial configuration* is composed of five security hosts, ten policy hosts, ten coordination hosts, five application hosts, and two DB application hosts. *Cfg. 1* refers to the increase by one unit of DB application hosts, whereas *Cfg. 2* indicates the last assignment of resources computed, i.e., the increase of five security hosts and of 2 DB application hosts. Finally, *Cfg. 3* refers to the configuration, which cannot be afforded with such a budget (\$20 000): an increase in the security hosts by six units, the DB application hosts by three units, and the application hosts by one unit with respect to the initial configuration. As can be observed in Fig. 8, the cost of the last resources configuration

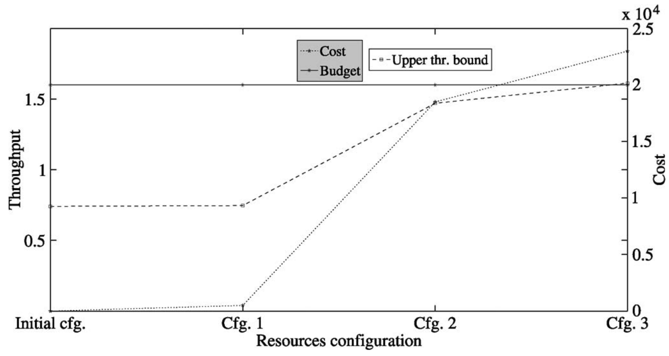


Fig. 8. Different resource configurations and their associated cost.

exceeds the assigned budget; therefore, the solution for the resource distribution is the previous configuration.

The evolution of the upper throughput bound is worth remarking. With the initial configuration, the upper throughput bound is $\Theta = 0.740740$. In the first configuration, the upper throughput bound increases by 0.75% ($\Theta = 0.746271$), whereas in the second configuration, it increases by almost 100% ($\Theta = 1.470598$). Finally, with the third configuration, the upper throughput bound increases by 9.68% ($\Theta = 1.612920$).

VII. Conclusion

The formalism of PNs allows one to model the behavior of a large class of artificial systems in which resources are shared by the different tasks. The performance of these systems, which is usually measured as the number of completed operations per time unit, is often a system requirement. Unfortunately, in most cases of interest, it is not possible to compute the exact performance of a system in a reasonable time due to the state explosion problem inherent to large discrete systems. Thus, the explosion problem poses difficulties not only for computing exactly the performance of an existing system but also for correctly designing new systems.

This paper has focused on the class of process PNs, which allows a wide variety of modeling possibilities while offering interesting analysis properties. For this class of nets, two iterative strategies have been proposed. The first aims at estimating efficiently the performance of a given system. Such an estimation is carried out by computing increasingly larger system bottlenecks. The goal of the second strategy is, given an initial budget and a cost of each resource, to gauge the number of instances of each resource so that the system performance is maximized, and the budget is not exceeded. This has been achieved by exploiting the linear dependence of the performance bounds with respect to the number of resources.

Given that both techniques make intensive use of LP techniques and the number of required iterations is usually low, their complexity and computational time are also low. The proposed strategies have been applied to a process PN modeling an SDBS. The performance of such a system has been evaluated for different workloads, and a distribution of resources that maximizes the throughput for a given budget has been estimated. We have developed a tool, PeabraiN [31], which implements the strategies here presented to make their use easier for practitioners. It enables both performance estimation and resource optimization to be computed in systems modeled with PNs. As future work, we plan to research into the quality of the initial upper bound obtained and to extend both strategies to more general PN classes.

Appendix

The strict inequality $\sum_{p\in V}\mathbf{y}(p) > 0$ in (10) is used to compel the components of places that belong to the next slowest p-semiflow to be positive. Once the LPP (10) is solved, only the strictly positive components are selected. When the solver precision is not very high, zero components might not be distinguishable from positive components with low values. To avoid this, $\sum_{p\in V}\mathbf{y}(p) > 0$ is replaced by $\sum_{p\in V}\mathbf{y}(p) > h$, with a strictly positive h. Thus, we need to find a value $h > 0$ that retains the feasibility of constraints $\mathbf{y}\cdot\mathbf{C} = \mathbf{0}$, $\mathbf{y}\cdot\mathbf{m_0} = 1$. A possible value h, such that $\mathbf{y} \geq h\cdot\mathbf{1}$, and fulfills both equations, can be calculated in the following way.

Recall that by the process PN structure, the number of p-semiflows is equal to $n+1$, where $n = |P_R|$ is the number of resources in the process PN system. Note also that the initial marking $\mathbf{m_0}$ of the system will be $\mathbf{m_0}(p) > 0,\ \forall p \in P_R \cup P_0$, $\mathbf{m_0}(p) = 0,\ \forall p \in P_S$. A p-semiflow $\mathbf{y}$ that covers all places can be computed by a linear combination of all minimal p-semiflows. Remember that each resource has an associated minimal p-semiflow (Property 3 of Definition 3).

Let us consider a system with n resources. Then, a linear combination of all minimal p-semiflows is $\mathbf{y} = \alpha_1\cdot\mathbf{y}_1 + \alpha_2\cdot\mathbf{y}_2 + \cdots + \alpha_{n+1}\cdot\mathbf{y}_{n+1},\ \ \alpha_i > 0, \forall i \in \{1\ldots(n+1)\}$. As $\mathbf{y}$ is a linear combination of p-semiflows, then $\mathbf{y}\cdot\mathbf{C} = \mathbf{0}$ is fulfilled. However, factors α_i must be adjusted in order to properly fulfill equation $\mathbf{y}\cdot\mathbf{m_0} = 1$. An intuitive idea for doing this is the following: As $\mathbf{y}(p)\cdot\mathbf{m_0}(p) > 0 \Leftrightarrow p \in P_R \cup P_0$, then $\mathbf{y}\cdot\mathbf{m_0} = 1$ can be reformulated as $\alpha_1\cdot\mathbf{y}_1(p_{r_1})\cdot\mathbf{m_0}(p_{r_1}) + \alpha_2\cdot\mathbf{y}_2(p_{r_2})\cdot\mathbf{m_0}(p_{r_2}) + \cdots + \alpha_{n+1}\cdot\mathbf{y}_{n+1}(p_{r_{n+1}})\cdot\mathbf{m_0}(p_{r_{n+1}}) = 1$, where p_{r_i} represents the place associated to resource $r_i,\ \forall i \in \{1\ldots n\}$, and $p_{r_{n+1}}$ is the process-idle place.

By the process PN structure, all positive values of $\mathbf{y}_i$ will be equal to 1. Therefore, the values α_i that fulfill the equation $\mathbf{y}\cdot\mathbf{m_0} = 1$ can be easily calculated as

$$\alpha_i = \frac{1}{\mathbf{m_0}(p_{r_i})\cdot(n+1)}\ \forall i \in \{1\ldots(n+1)\}\,.$$

Hence, a possible value h that fulfills $\mathbf{y}(p) \geq h,\ \forall p \in P$ is, in this case, $h = \min(\alpha_i),\ \forall i \in \{1\ldots(n+1)\}$. Such a value relates the number of resources in the system and the number of resource instances. Thus, the value of h for most systems of interest in practice is much higher than the numerical tolerance of the LPP solver (in this paper, the numerical tolerance of the LPP solver has been set to 10^{-5}).

As the objective function in LPP (11) is maximized, the value h obtained from that LPP will be at least equal to $\min(\alpha_i)\ \forall i \in \{1\ldots(n+1)\}$, i.e., $h \geq \min(\alpha_i)\ \forall i \in \{1\ldots(n+1)\}$.

References

[1] J. Colom, "The resource allocation problem in flexible manufacturing systems," in *Applications and Theory of Petri Nets*, W. van der Aalst and E. Best, Eds. Berlin, Germany: Springer-Verlag, 2003, ser. LNCS, pp. 23–35.

[2] F. Tricas, "Deadlock analysis, prevention and avoidance in sequential resource allocation systems," Ph.D. dissertation, Universidad de Zaragoza, Dpto. de Informática e Ingeniería de Sistemas, Zaragoza, Spain, May, 2003.

[3] F. Tricas, F. Vallés, J. Colom, and J. Ezpeleta, "An iterative method for deadlock prevention in FMS," in *Discrete Event Systems. Analysis and Control*, R. Boel and G. Stremersch, Eds. Boston, MA, USA: Kluwer, Aug. 2000, pp. 139–148.

[4] J. Ezpeleta and R. Valk, "A polynomial deadlock avoidance method for a class of nonsequential resource allocation systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 36, no. 6, pp. 1234–1243, Nov. 2006.

[5] N. Wu, M. Zhou, and Z. Li, "Resource-oriented petri net for deadlock avoidance in flexible assembly systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 1, pp. 56–69, Jan. 2008.

[6] J. Lopez-Grao and J. Colom, "On the deadlock analysis of multithreaded control software," in *Proc. IEEE 16th Conf. ETFA*, Sep. 2011, pp. 1–8.

[7] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on petri nets—A literature review," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 437–462, Jul. 2012.

[8] F. Tricas and J. Ezpeleta, "Computing minimal siphons in petri net models of resource allocation systems: A parallel solution," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 36, no. 3, pp. 532–539, May 2006.

[9] S. Wang, C. Wang, M. Zhou, and Z. Li, "A method to compute strict minimal siphons in a class of petri nets based on loop resource subsets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 1, pp. 226–237, Jan. 2012.

[10] G. Chiola, C. Anglano, J. Campos, J. Colom, and M. Silva, "Operational analysis of timed petri nets and application to the computation of performance bounds," in *Proc. 5th Int. Workshop PNPM*, Toulouse, France, Oct. 1993, pp. 128–137.

[11] J. Campos, G. Chiola, J. Colom, and M. Silva, "Properties and performance bounds for timed marked graphs," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 39, no. 5, pp. 386–401, May 1992.

[12] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling With Generalized Stochastic Petri Nets*. Hoboken, NJ, USA: Wiley, 1995, ser. Wiley Series in Parallel Computing.

[13] C. Ramchandani, "Analysis of asynchronous concurrent systems by petri nets," Ph.D. dissertation, Dept. of Electrical Engineering, Massachusetts Inst. Technol., Cambridge, MA, USA, Feb. 1974.

[14] R. J. Rodríguez and J. Júlvez, "Accurate performance estimation for stochastic marked graphs by bottleneck regrowing," in *Proc. 7th EPEW*, vol. 6342, *LNCS*, Sep. 2010, pp. 175–190.

[15] Z. Liu, "Performance bounds for stochastic timed petri nets," in *Proc. 16th ICATPN*, 1995, pp. 316–334.

[16] S. Haddad, P. Moreaux, M. Sereno, and M. Silva, "Product-form and stochastic Petri nets: A structural approach," *Perform. Eval.*, vol. 59, no. 4, pp. 313–336, Mar. 2005.

[17] G. Casale, N. Mi, and E. Smirni, "Bound analysis of closed queueing networks with workload burstiness," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 1, pp. 13–24, Jun. 2008.

[18] T. Osogami and R. Raymond, "Semidefinite optimization for transient analysis of queues," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 1, pp. 363–364, Jun. 2010.

[19] J. Li, Y. Fan, and M. Zhou, "Performance modeling and analysis of workflow," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 2, pp. 229–242, Mar. 2004.

[20] H. Wang and Q. Zeng, "Modeling and analysis for workflow constrained by resources and nondetermined time: An approach based on petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 4, pp. 802–817, Jul. 2008.

[21] K. V. Hee, H. Reijers, E. Verbeek, and L. Zerguini, "On the optimal allocation of resources in stochastic workflow nets," in *Proc. 7th UK Perform. Eng. Workshop*, K. Djemame and M. Kara, Eds., Leeds, U.K., 2001, pp. 23–34.

[22] Y.-L. Chen, P.-Y. Hsu, and Y.-B. Chang, "A petri net approach to support resource assignment in project management," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 3, pp. 564–574, May 2008.

[23] J. Colom, J. Campos, and M. Silva, "On liveness analysis through linear algebraic techniques," presented at the Annu. Gen. Meet. ESPRIT Basic Res. Action Des. Methods Based on Nets (DEMON), Paris, France, Jun. 1990.

[24] H. Hu, M. Zhou, and Z. Li, "Liveness and ratio-enforcing supervision of automated manufacturing systems using petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 2, pp. 392–403, Mar. 2012.

[25] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[26] G. Florin and S. Natkin, "Necessary and sufficient ergodicity condition for open synchronized queueing networks," *IEEE Trans. Softw. Eng.*, vol. 15, no. 4, pp. 367–380, Apr. 1989.

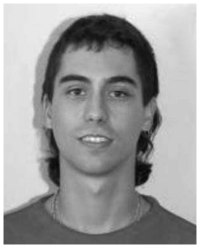
[27] J. Campos and M. Silva, "Structural techniques and performance bounds of stochastic petri net models," in *Proc. Adv. Petri Nets*, vol. 609, *Lecture Notes in Computer Science*, 1992, pp. 352–391.

[28] J. D. C. Little, "A proof for the queuing formula: $L = \lambda W$," *Oper. Res.*, vol. 9, no. 3, pp. 383–387, May/Jun. 1961.

[29] E. M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*. Great Barrington, MA, USA: North River, 1986.

[30] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. New York, NY, USA: Springer-Verlag, 2004.

[31] R. J. Rodríguez, J. Júlvez, and J. Merseguer, "PeabraiN: A PIPE extension for performance estimation and resource optimisation," in *Proc. 12th Int. Conf. ACSD*, 2012, pp. 142–147.



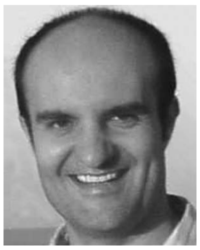
Ricardo J. Rodríguez received the B.S. and M.S. degrees in computer science engineering from the University of Zaragoza, Zaragoza, Spain, in 2008 and 2010, respectively. He is currently working toward the Ph.D. degree with the Department of Computer Science and Systems Engineering, University of Zaragoza.

He was a Visiting Researcher with the School of Computer Science and Informatics, Cardiff University, Cardiff, U.K. He is currently a Research Assistant with the Department of Computer Science and Systems Engineering, University of Zaragoza. His research interests include performance analysis and optimization of large discrete event systems, secure software engineering, and fault-tolerant systems.

Mr. Rodríguez is a member of the Aragon Institute of Engineering Research, and IEEE Computer Society. He has been serving as a Referee for international journals and for several international conferences and workshops.

Jorge Júlvez received the M.S. and Ph.D. degrees in computer science engineering from the University of Zaragoza, Zaragoza, Spain, in 1998 and 2005, respectively. His Ph.D. study was related to the study of qualitative and quantitative properties of continuous Petri nets.

In 2005, he joined as a Postdoctoral Researcher with the Department of Software, Technical University of Catalonia, Catalonia, Spain, where he has spent three years. As a Researcher, he visited the Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari, Italy; the Department of Information Engineering, University of Siena, Siena, Italy; the SYSTeMS Research Group, University of Ghent, Ghent, Belgium; and the Computing Laboratory, University of Oxford, Oxford, U.K. Since 2008, he has been with the Department of Computer Science and Systems Engineering, University of Zaragoza, where he is currently an Associate Professor. His current research interests include analysis and optimization of large discrete event systems.



José Merseguer received the B.S. and M.S. degrees in computer science and software engineering from the Technical University of Valencia, Valencia, Spain, and the Ph.D. degree in computer science from the University of Zaragoza, Zaragoza, Spain.

He has been a Visiting Researcher with the University of Turin, Turin, Italy, and with the University of Cagliari, Cagliari, Italy. He is currently a Director with the Department of Computer Science and Systems Engineering, University of Zaragoza. He teaches software engineering courses at graduate and undergraduate levels. He has developed postdoctoral research with Prof. M. Woodside at Carleton University, Ottawa, ON, Canada, and with Prof. R. Lutz at Iowa State University, Ames, IA, USA. His main research interests include performance and dependability analysis of software systems, Unified Modeling Language semantics, and service-oriented software engineering.

Dr. Merseguer is also a member of the Aragon Institute of Engineering Research. He has been serving as a Referee for international journals and as a Program Committee member for several international conferences and workshops.