Chapter 10

Modeling and Simulation of Task Allocation with Colored Petri Nets

Mildreth Alcaraz-Mejia,
Raul Campos-Rodriguez and
Marco Caballero-Gutierrez

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/67950

Abstract

The task allocation problem is a key element in the solution of several applications from different engineering fields. With the explosion of the amount of information produced by the today Internet-connected solutions, scheduling techniques for the allocation of tasks relying on grids, clusters of computers, or in the cloud computing, is at the core of efficient solutions. The task allocation is an important problem within some branch of the computer sciences and operations research, where it is usually modeled as an optimization of a combinatorial problem with the inconvenience of a state explosion problem. This chapter proposes the modeling of the task allocation problem by the use of Colored Petri nets. The proposed methodology allows the construction of compact models for task scheduling problems. Moreover, a simulation process is possible within the constructed model, which allows the study of some performance aspects of the task allocation problem before any implementation stage.

Keywords: colored Petri nets, task allocation problem, modeling and simulation, distributed computing

1. Introduction

The services provided by Internet-based modern applications require the execution of massively parallel processes in order to produce time-effective information for different requirements. The processing of these "Big Data" is very computing, demanding in almost all of the cases. Thus, vast server farms installed over the world are ready to process the requests from different users. The digital data available in the web are huge and diverse, therefore complex, and in a continuous growth rate.

Nowadays, most of the queries on Internet are produced from smartphones and personal computers. However, the data traffic on web is expected to grow in an exponential rate, as the technology related to the Internet of Things (IoT) allows the connection of other devices from the houses, smart warehouses, automated driving cars, machinery from the agriculture, UAV's for goods delivering, or smart cities, to mention a few. In order to cope with the challenges related to a massive number of connected devices, new ways of computing emerged. Most of them are based on parallel and distributed computing, such as clusters, grids, and cloud computing paradigms.

At the core of these paradigms are the tasks scheduling and resources allocation problems. Adaptations of theory, methods, algorithms, tools, and others are arising in order to cope with the increasing number of cores per processor, as well as with the interconnected and heterogeneous computers, considering that, in many cases, those computing architectures are connected through the Internet. The tasks scheduling and resources allocation were well studied in the context of architectures with a relatively small number of processors. One of the fundamental concerns deals with designing algorithms for an efficiently allocation of the tasks among all the available processes and resources.

The answer to this question may lead to a hard work, since, in the worst case, it involves an exponential number of possible solutions [1]. Thus, a simulation process, for scenarios with a big number of computing elements, or cores, as in cloud and grid computing, is a valuable tool. In this context, the modeling and simulation are disciplines widely used to obtain feedback and statistics information, to improve associated applications and algorithms. Through experimentation in simulators, a fast prototyping process allows performance evaluation and parameter tuning for different workload conditions. To perform an efficient simulation process, a suitable modeling method is required. The Petri nets (PN's) and its extension, such as colored PN (CPN), have been considered as an efficient modeling technique for concurrent and distributed systems.

The simulation and implementation stages of scheduling and resource allocation problems in distributed systems have been successfully addressed [2–9]. Moreover, variants of PN's are used for modeling and simulation of other distributed concepts related to cloud services and computing [9–16], as well as in the field of the diagnosis of discrete event systems [17–19], reconfiguration [20–24], fuzzy inference [25] or identification [26].

This work proposes the modeling of the task allocation problem by the use of a methodology based on CPN's. It also shows how to simulate these models in order to obtain relevant information for the design process. To illustrate the techniques, this work considers a tree structure and supposes that the set of tasks and the set of processes are fixed and defined before the task allocation is performed as in Ref. [27]. The proposed CPN modeling method represents the behavior of a set of processors, or working threads, that traverse a tree structure from the root to a single leaf in order to acquire a task. The processes decide what route to take at each node of the tree structure, by using a decision function. Then, it routes the tree backwards updating some global variables that serves as inter-process communication mechanism.

The proposed method allows obtaining a CPN, which is a compact representation of the problem, and at the same time, it allows simulating its behavior and obtaining performance

http://dx.doi.org/10.5772/67950

graphics. The colored tokens represent the processes, tasks, and other variables of interest in the problem. A decision function is attached to some transitions of the CPN model. This function influences the behavior of the process or thread. Due to the graphical nature of the CPN, in addition to the mathematical fundamentals, this methodology provides a great framework to simulate and analyze the task allocation problem, avoiding the state space explosion, and thus, having a compact representation of a complex behavior. The flexibility of the modeling framework allows simulating different scenarios, using different decision functions, as well as varying the number of initial processes or threads. Additionally, the Petri Net model can be easily extended to simulate a wider number of tasks.

The rest of this chapter includes a background of the task allocation and several techniques that have been used for addressing this problem. The modeling framework based on colored Petri nets is next presented and the modeling methodology is detailed. This methodology highlights the main aspects of the task allocation problem, such as the concepts of task and process, as well as the balancing policies and how to capture them by CPN blocks. The simulation of the obtained CPN models is then presented and discussed. Some performance charts are provided. The charts allow the study of key aspects of task allocation problem such as the effective work done by a processor, the task contention, the effects of different balancing policies, to mention a few. With this information, the scientists and engineers are able to study different aspects of the problem prior any implementation stage in a particular field of interest.

2. Task allocation problem

Task allocation problem has been addressed for several purposes and using different techniques. Analytic solutions are in some cases prohibitive since they require the analysis of an exponential number of possible solutions. Thus, some designs use probabilistic approaches to produce good solutions in a reasonable time. Jevtić et al. [28] present a distributed bee's algorithm for static task allocation inspired on the behavior of a colony of bees and based on the optimization of a cost function. They also show results using the Arena simulator. Delle Fave et al. [29] introduced a decentralized optimization for static task assignment based on a constrained utility function, firstly evaluated by simulations. Johnson et al. [30] state a distributed multi-agent algorithm for static multi-task allocation as an optimization problem based on mixed-integer programming. They also provide some results of Monte Carlo simulation. Zhao-Pin et al. [31] introduced a distributed and static task allocation algorithm for multi-agents that learn how to determine by themselves what tasks should realize and how to form coalitions for cooperation, via their proposed profit-sharing learning mechanism. Macarthur et al. [32] introduced a dynamic and distributed algorithm for multi-agent task allocation problems, based on fast-max-sum algorithm combined with a branch-and-bound technique, in order to reduce the execution time in the allocation of task to its respective agent. Alistarh et al. [27, 33] present a decentralized task allocation, static and dynamic, based on to-do trees, where decisions on every inner node are based on a probability function.

One of these most used techniques, due to its simplicity of implementation, is based on a rooted tree structure that the process traverses in order to acquire a task [7, 8]. The tree structure allows to the working processes for executing a non-deterministic behavior, which

is quite suitable for real environments. Typically, in these tree structures, every reference to a task is allocated on the leaves. Thus, every internal vertex allows to a working process taking a decision based on a function over the number of tasks in every leaf of the subtrees from each child of the current vertex. Hence, the decision function, which could be a simple deterministic algorithm, or a sophisticated statistical or stochastic procedure, is a key element for which a process traverses through the tree structure, in this kind of task allocation techniques.

Figure 1 shows a binary tree that represents a scheduling problem where eight tasks have to be attended. The height of the tree is $H = \log_2(8) = 3$, where the level of $v_{2,2}$ is 2, the vertex $v_{1,0}$ is said to be parent, or ancestor, of $v_{2,0}$ and $v_{2,1}$, and the latter two are said to be children, or descendant, of $v_{1,0}$. The vertex $v_{0,0}$ is the root, and every vertex is internal except for all of the level 3, i.e., $v_{3,x}$ with $x \in [0, 7]$, which are known as leafs of the tree. The tree formed by the vertexes $v_{2,1}$, $v_{3,2}$, $v_{3,3}$ and the corresponding edges that connects them, is a subtree with $v_{2,1}$ as a root.

From the graph theory point of view, a DT is an acyclic graph in which a unique path connects every node to others, represented in a top-to-bottom fashion, where the root, i.e., initial node, is plotted at the top of the tree. The DT is a binary decision tree (BDT) if at every node is considered at most two possible decisions, as shown in **Figure 1**. For more information on trees see Ref. [34].

2.1. Task scheduling with DT

By using a tree structure like the one depicted in **Figure 1**, the task allocation algorithm is relatively simple. Each working process shares the tree structure and iterates in an infinite loop, trying to acquire a new task to execute, which is located at the leafs of the tree. Each working process starts at the root node of the tree and decides whether to go to the left or the right child node. The process knows an approximation of the total pending tasks at the left and right subtrees at every step. For example, in **Figure 1**, each working thread knows that at the beginning of the scheduling process, at the root node, that there exist four pending tasks at the

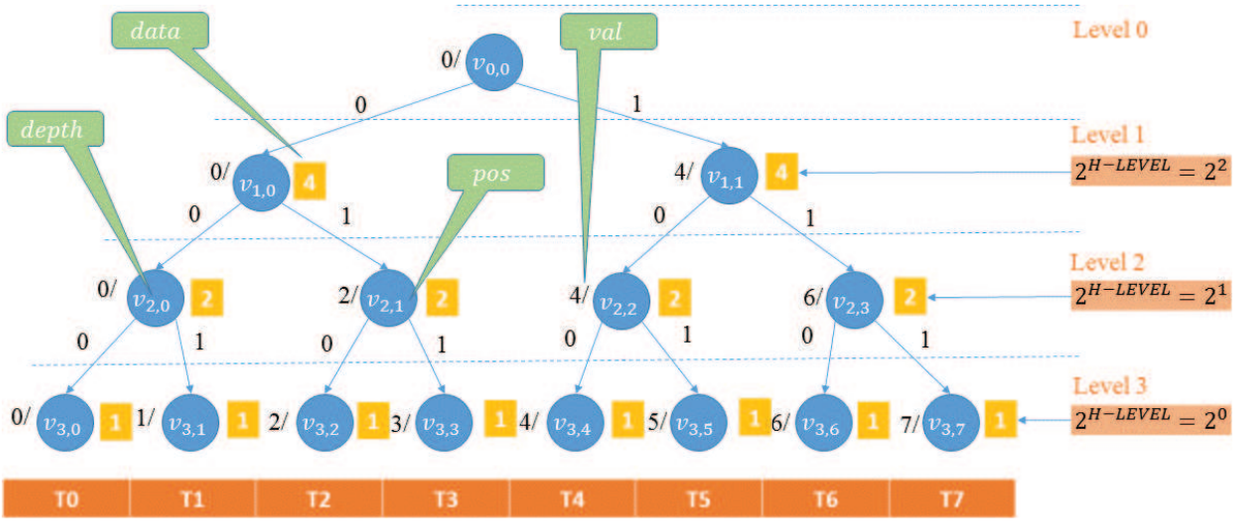

Figure 1. A direct rooted full binary tree.

http://dx.doi.org/10.5772/67950

left subtree and four pending tasks at the right one. Then, at every stage, the processes decide to traverse to the left or right in an asynchronous fashion. The total number of pending tasks is backward updated from the lower nodes to the uppers as the tasks are successfully executed by the threads. Notice that, the number of pending tasks at the left or right subtree is an approximation since this number may be not updated yet.

Since the processes are asynchronous among them, it is possible that they arrive to the leaf nodes that may occur at different rates. Every time that a process reaches a leaf, it asks for the execution of the task. If such a task is free, then the process blocks this task, executes the activity, and marks the task as executed. Then, it goes back to the root node to seek for a new task to execute, while the information of the remaining tasks is backward updated from the leaf to the root node. It is possible that when a process reaches a leaf node, the corresponding task was already executed by another process. In this case, the process goes back to the root node, and this miss is counted to measure the effectiveness of the algorithms.

$$r < \frac{x}{x+y} \tag{1}$$

In Eq. (1), it represented a decision rule that a working process may execute at every level of the BDT. Let x be the total number of pending tasks in the left subtree and y be the total number of pending tasks in the right subtree, while r is a random number. Thus, the probability to go to the left subtree in a flip coin is given by r. As greater the ratio, it is greater the probability to go to the left child of the current node. One of the main objectives of this work is to measure the effectiveness of a decision rule as Eq. (1) by means of a simulation process.

3. Colored Petri nets

PN's are a modeling framework that combines a graphical visualization with a mathematical model [35]. The CPN's are one of the extensions to PN's [9, 17, 35–42], which allow the construction of compact representations of big models. In this section, basic notions of CPN are presented.

The main characteristic that differentiates CPN from other type of nets resides in the token definition. In a CPN, the tokens can stand for complex data besides of a single value, such as an integer, a real, Boolean or strings. This characteristic allows for the representation of elaborated data types, similar to those used by high-level programming languages. This ability exploits the multi-set cardinality to construct compact models that otherwise are in the power set of the colored tokens. The formal definition of a CPN is as follows [35].

Definition 1. A Colored Petri Net (CPN) is a tuple

$\mathcal{N} = \langle P, T, Pre, Post, B, F, Cp, Ct \rangle$ where its elements are described as:

- P is a finite set of places of $\mathcal{N}$, with $m = |P|$,
- T is a finite set of transitions of $\mathcal{N}$, such that $T \cap P = \varnothing$, with $n = |T|$,

- B is finite set of color classes,
- F is a set of conditions,
- $C_p : P \rightarrow B$ is the color domain mapping,
- $C_t : T \rightarrow F$ is a conditional color mapping, and
- $Pre, Post \in S^{|P|\times|T|}$ are matrices representing the input and output incidence matrices of $\mathcal{N}$, such that $Pre[p, t] : C_t(t) \rightarrow Bag(C_p(p))$ and $Post[p, t] : C_t(t) \rightarrow Bag(C_p(p))$ are mappings for every pair $(p, t) \in P \times T$; where $Bag(S) = \cup\, b(s_i)$ for all $s_i \in S$, such that $b(s_i) = \sum s_i \in S$.

The incidence matrix is $C = Post - Pre$. The mapping given by $Pre[p, t] : C_t(t) \rightarrow Bag(C_p(p))$, defines for each conditional color mapping f of t (*i.e.* $\forall f_i \in C_t(t)$), the token bag to be removed from p, in the occurrence of t, under color condition f. In the same way, $Post[p, t](f)$ specifies the token bag to be added to p, in the occurrence of t, under color condition f.

Definition 2. A marking M of a CPN $\mathcal{N}$ is a vector of size $m = |P|$, such that $M(p) \in Bag(C_p(p))$ for every $p \in P$. The vector M_0 denotes the initial marking of the net $\mathcal{N}$ and the pair $(\mathcal{N}, M_0)$ is known as a CPN System.

Definition 3. A transition in a CPN System $(\mathcal{N}, M_0)$ is said to be enabled for a color condition f in a marking M iff $M \geq Pre[\blacksquare, t](f)$, denoted as $M \xrightarrow{t,f}$.

Definition 4. The marking evolution w.r.t a color condition f is given by $M' = M + Post[\blacksquare, t](f) - Pre[\blacksquare, t](f) = M + C[\blacksquare, t](f)$, denoted by $M \xrightarrow{t,f} M'$. For a general color condition, it is denoted as $M \xrightarrow{t} M'$ where f is implicit in a context.

The net in **Figure 2** represents a very simple CPN System $(\mathcal{N}, M_0)$ where:

- $P = \{p_1, p_2, p_3\}$,
- $T = \{t_1, t_2\}$,
- $B = o \cup r$, where $o = \{o_1, o_2\}$ and $r = \{r_1, r_2\}$ are variables,
- $F = \{f_1, f_2\}$,

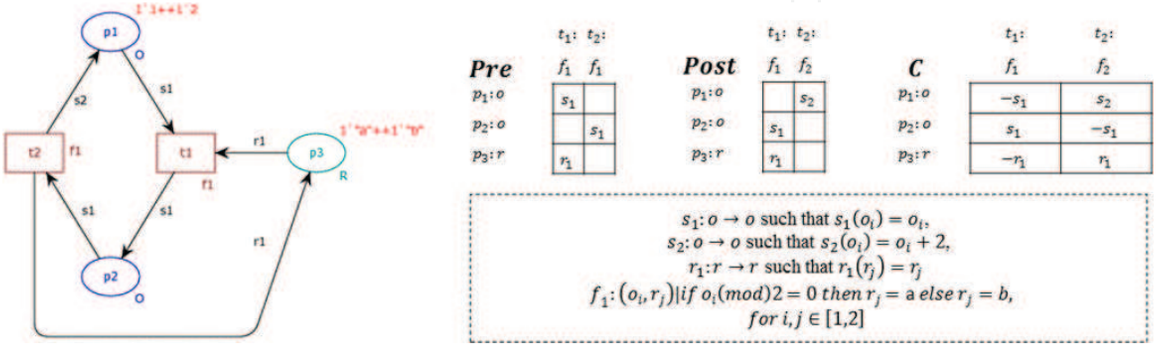

Figure 2. A CPN example.

- $C_p(p_1) = C_p(p_2) = o,\ C_p(p_3) = r$,
- $C_t(t_1) = f_1$, $C_t(t_2) = f_2$,
- *Pre* and *Post* are as depicted, and
- $M_0 = [\{o_1, o_2\}, \varnothing, \{r_1, r_2\}]'$, where $o_1 = 1$, $o_2 = 2$, $r_1 = a$, $r_2 = b$.

Notice that, at the initial condition, $M_0 \geq Pre[\blacksquare, t_1](f_1)$. That is $[\{o_1, o_2\}, \varnothing, \{r_1, r_2\}]' \geq [\{o_1, o_2\}, \varnothing, \{r_1, r_2\}] : \{(o_1, r_1)(o_2, r_2)\}$, thus, t_1 can be fired for any color binding $\{(o_1, r_2)(o_2, r_1)\}$ due to the condition in f_1. Suppose that, t_1 fires for color condition f_1 with (o_2, r_1), then $M_0 \xrightarrow{t_1, (o_2, r_1)} M_1$, with $M_1 = [\{o_1\}, \{o_2\}, \{r_2\}]'$, where $o_1 = 1$, $o_2 = 2$, $r_2 = b$, by $Post[\blacksquare, t_1](f_1) = \{(s_1)\}$ as detailed in the figure. Now, $M_1 \xrightarrow{t_1, (o_1, r_2)}$ and $M_1 \xrightarrow{t_2, (o_2)}$, in such a way, that if t_2 is fired, then $M_1 \xrightarrow{t_2, (o_2)} M_2$, with $M_2 = [\{o_1, o_2\}, \varnothing, \{r_1, r_2\}]'$, where $o_1 = 1$, $o_2 = 4$, $r_1 = a$, $r_2 = b$. However, if t_1 is fired from M_1 for the color condition (o_1, r_2), then $M_1 \xrightarrow{t_1, (o_1, r_2)} M_3$, with $M_3 = [\varnothing, \{o_1, o_2\}, \varnothing]'$, where $o_1 = 1$, $o_2 = 2$.

Roughly speaking, the CPN binds the even number in p_1 to the character "a" in p_3 and the odd number to character "b." This is defined by the guard function f_1 attached to t_1. The function s_1 discards the r element of the token bound to the firing of t_1, while the firing of t_2 adds two to the o element of the token and recovers the r element that is put back to the place p_3. Thus, the tones in p_1 changes to $o_1 = 1, 3, 5, \ldots$ and $o_2 = 2, 4, 6, \ldots$, while the characters "a" and "b" in p_3 remain the same all the time. The "R" in the place p_3 stands for "resources," which is the function intended for the characters in this CPN model.

Notice that, the bindings (o_1, r_1) and (o_2, r_2) are discarded by the function f_1. This is one of the advantages of the CPN modeling tool, since otherwise, the combinatorial explosion of possible bindings turns untreatable under some circumstances that typically arises in real applications. For more information about the CPN modeling formalism and related analysis and tools, see Ref. [36].

4. Modeling and simulation of task scheduling

The construction of a generalized model for a BDT based on CPN considers d different features and nT classes. The procedure includes the identification of the transitions and places of the model, the arcs and its labeling, as well as the multi-set of colored tokens. The remaining of this section is devoted to detail the methodology and illustrate it by short examples.

4.1. Structure of the task allocation as a CPN

Consider the following steps for the construction of a CPN model that captures the structure of a BDT. It is supposed that the BDT is a full binary tree with a complete set of tasks represented by a structure called Task Array. Thus, the size of the Task Array is 2^H, where the height of the tree is H.

Step 1. Labeling nodes in the BDT

Firstly, suppose that the DT is a full binary tree. Then, assign "0" to every left edge and "1" to every right edge. This labeling represents the result of the flip coin in the task allocation procedure of a BDT discussed in the previous section.

After that, some information is used to label every node with a pair $(depth, position)$, where *depth* is the level of the node, and *position* is the corresponding position from left to right. For example, consider the root node $(0, 0)$ in the section of the BDT depicted in **Figure 3**. It is a fraction of the tree of height $H = 3$ of **Figure 1**. Then, to label the children of the root node proceed as follows:

- Let be $depth = depth - of - the - parent + 1$ and $position = 2 * position - of - the - parent + h$, with $h = 0$ if this node is the left child, $h = 1$ otherwise.

Therefore, the label of the left child is $(depth = 0 + 1, position = 0 * 2 + 0) = (1, 0)$, while the label of the right child is $(depth = 0 + 1, position = 0 * 2 + 1) = (1, 1)$.

Step 2. Building the CPN System

Let $(\mathcal{N}, M_0)$ be a CPN System for a dynamic binary decision tree for nT classes and d features with $\mathcal{N} = \langle P, T, Pre, Post, B, F, C_p, C_t \rangle$ constructed as follows:

- $P = P_O \cup P_R \cup P_A$, where P_O is the set of places with assigned tokens of type O, P_R is the set of places with assigned tokens of type R, P_A is the place with assigned tokens of type A. $|P_O| = 2(H + 1)$, $|P_R| = H$ and $|P_A| = 1$, where $H = \log_2 nT$. Places of type O represent every level from the root to the leaves, forward, and backwards, i.e., a token in one of those place contains the information of *pos*, *depth*, *val*. Places of type R represent the state of every subtree from every node at that level, i.e., the value of the *data* function for every node of the same level given by *pos* and *depth*. Place of type A represent the *Task Array* state, i.e., the information of the availability of the task.
- T is the set of transitions with $|T| = |P_O|$.

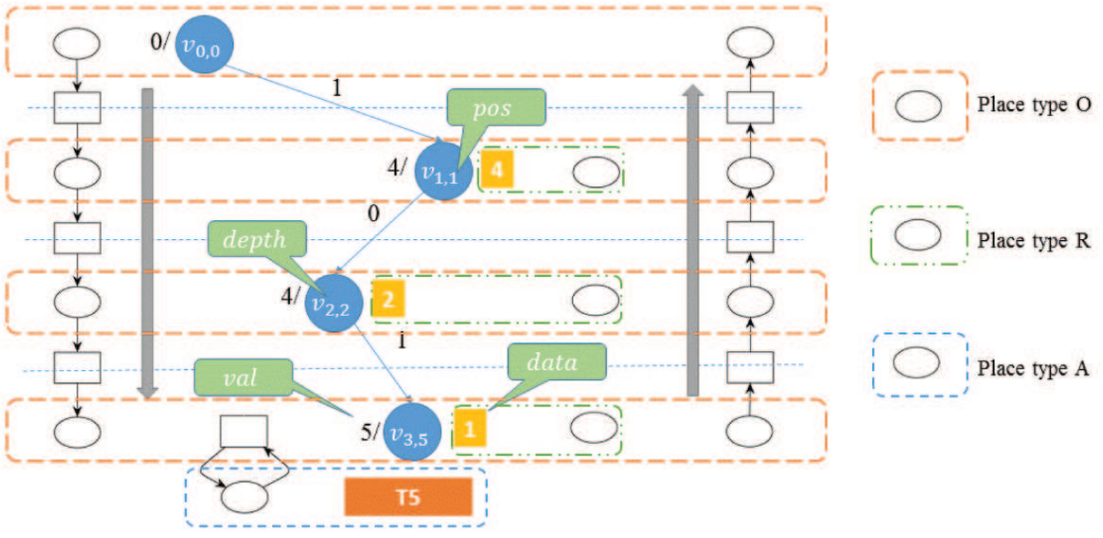

Figure 3. Relation BDT and CPN.

http://dx.doi.org/10.5772/67950

- $B = o \cup r \cup a$, for $o = \cup o_i$, $r = \cup r_j$, and $a = \cup a_l$ where $o_i = \langle depth_i, pos_i, val_i, success_i \rangle$, $r_j = \langle depth_j, pos_j, data_j \rangle$, $a_l = \langle pos_l, ready_l \rangle$ for $i \in [1, d]$, $j \in [1, 2^{H+1} - 2]$, and $l \in [1, nT]$, with $depth_i$, pos_i, val_i, $success_i$, $data_j$, $exist_l$ variables and $depth_j$, pos_j, pos_l constants.

- $F = \{f_1, f_2, f_3, f_4\}$, such that $f_1 : (o_i, r_j, r_k) | depth_i + 1 = depth_j = depth_k \wedge pos_i * 2 = pos_j \wedge pos_i * 2 + 1 = pos_k$, $f_2 : (o_i, r_j) | depth_i = depth_j \wedge pos_i = pos_j$, $f_3 : (o_i, a_j) | pos_i = pos_j$, $f_4 : (o_j)$.

- $C_p(p_i) = o$ for all $i \in [1, |P_O|]$, $C_p(p_i) = r$ for all $i \in [|P_O| + 1, |P|]$ and $C_p(p_i) = a$ for $i = 2(H + 1) + H + 1$.

- $C_t(t_i) = f_1$ for $i \in [1, H]$, $C_t(t_i) = f_2$ for $i \in [H + 2, |P_O| - 1]$, $C_t(t_i) = f_3$ for $i = H + 1$ and $i = |P_O| + 1$.

- $Pre[p_i, t_i](f_1) = si_1 | f_1$, for $i \in [1, H]$, $Pre[p_i, t_i](f_2) = si_1 | f_2$, for $i \in [H + 2, |P_O| - 1]$, $Pre[p_i, t_i](f_3) = si_1 | f_3$, for $i = H + 1$, $Pre[p_i, t_i](f_3) = si_1 | f_4$, for $i = |P_O| + 1$, $Pre[p_i, t_j](f_3) = qi_1 | f_3$, for $i = |P_O| + |P_R| + 1$, $j = H + 1$, $Pre[p_{u+i}, t_i](f_1) = 2ri_1 | f_1$, for $i \in [1, H]$, $Pre[p_{u+i}, t_{u-i}](f_2) = 2ri_2 | f_2$, for $i \in [1, H]$, where:

  - si_1: $o \rightarrow o$, such that $si_1(\langle depth_i, pos_i, val_i, success_i \rangle) = \langle depth_i, pos_i, val_i, success_i \rangle$.

  - qi_1: $a \rightarrow a$, such that $i_1(\langle pos_l, exist_l \rangle) = (\langle pos_l, exist_l \rangle)$.

  - ri_1: $r \times r \rightarrow r$, such that $ri_1(\langle depth_j, pos_j, data_j \rangle, \langle depth_k, pos_k, data_k \rangle) = \langle depth_j, pos_j, data_j \rangle \langle depth_k, pos_k, data_k \rangle$.

  - $ri_2 : r \rightarrow r$, such that $ri_2(\langle depth_j, pos_j, data_j \rangle) = \langle depth_j, pos_j, data_j \rangle$.

- $Post[p_{i+1}, t_i](f_1) = so_1 | f_1$, for $i \in [1, H]$, $Post[p_{i+1}, t_i](f_2) = so_2 | f_2$, for $i \in [H + 2, |P_O| - 1]$, $Post[p_{i+1}, t_i](f_3) = so_3 | f_3$, for $i = H + 1$, $Post[p_1, t_i](f_3) = so_4 | f_4$, for $i = |P_O|$, $Post[p_i, t_j](f_3) = qo_1 | f_3$, for $i = |P_O| + |P_R| + 1$, $j = H + 1 Post[p_{u+i}, t_i](f_1) = ro_1$, for $i \in [1, H]$, $Post[p_{u+i}, t_{u-i}](f_2) = ro_2$, for $i \in [1, H]$, where:

  - $so_1 : o \rightarrow o$, such that $so_1(\langle depth_i, pos_i, val_i, success_i \rangle) = \langle depth_i + 1, pos_i * 2 + h_i, val_i + (h_i) * 2^{H - depth_i + 1}, success_i \rangle$, $with h_i = 0$ *if* α *holds, otherwise* 1. α is a decision function.

  - $so_2 : o \rightarrow o$, such that $so_2(\langle depth_i, pos_i, val_i, success_i \rangle) = \langle depth_i - 1, \frac{pos_i}{2}, val_i, success_i \rangle$.

  - $so_3 : o \rightarrow o$, such that $so_3(\langle depth_i, pos_i, val_i, success_i \rangle) = \langle depth_i, pos_i, val_i, success_i = 1 if Exist(pos_i, a_l) = 1 \; otherwise \; 0 \rangle$, where $Exist(pos_i, a_l)$ returns 1 if for a color class $a_l = pos_l, exist_l$ with $pos_l = pos_i$, $exist_l = 1$, otherwise 0.

  - $so_4 : o \rightarrow o$, such that $so_4(\langle depth_i, pos_i, val_i, success_i \rangle) = \langle 0, 0, 0, 0 \rangle$,

  - qo_1: $a \rightarrow a$, such that $qo_1(\langle pos_l, exist_l \rangle) = (\langle pos_l, exist_l = 0 \; if Exist(pos_l, a_l) = 1, otherwise 1 \rangle)$.

  - ro_1: $r \times r \rightarrow r$, such that $ro_1(\langle depth_j, pos_j, data_j \rangle, \langle depth_k, pos_k, data_k \rangle) = \langle depth_j, pos_j, data_j \rangle \langle depth_k, pos_k, data_k \rangle$.

  - ro_2: $r \rightarrow r$, such that $ro_2(\langle depth_j, pos_j, data_j \rangle) = \langle depth_j, pos_j, data_j - 1 \; if success_i = 1, otherwise data_j \rangle$.

- The initial marking, assuming that the Task Array is initially full, is M_0 defined as follows:
  - $M_0[1] = \{o_1, \ldots, o_d\}$, where $o_i = depth_i = 0, pos_i = 0, val_i = 0, success_i = 0 = 0, 0, 0, 0$ for $i \in [1, d]$.
  - $M_0[|P_O| + i] = \cup_{i=1H} \cup_{j=02^i-1} r_{2^i-2+j} = \langle i, j, 2^{H-i} \rangle$ assuming that the Task Array is initially full, as mentioned.
  - $M_0[2(\mathrm{H}+1) + \mathrm{H} + 1] = \cup_{i=1nT} a_i$ where $a_i = \langle pos_i = i, exist_i = 1 \rangle$.
  - $M_0[\mathrm{i}] = \varnothing$ for $i \in [2, |P_O|]$.

Figure 4 shows the relation of the tree structure with the proposed CPN as stated by the previous definition. Observe that places of type O represent every level from the root to the leaves, forward, and backwards, i.e., a token in one of those place contains the information of *pos*, *depth*, and *val*. Places of type R represent the state of every subtree from every node at that level, i.e., the value of the *data* function for every node of the same level given by *pos* and *depth*. Place of type A represents the *Task Array* state, i.e., the information of the availability of the task.

4.2. Dynamics of the task allocation as a CPN

Typically, in these tree structures, every reference to a task is allocated on the leaves. Thus, every internal vertex allows to a working process taking a decision based on a function over the number of tasks in every leaf of the subtrees from each child of the current vertex. Hence, the decision function, which could be a simple deterministic algorithm, or a sophisticated statistical or stochastic procedure, is a key element for which a process traverses through the tree structure, in this kind of allocation techniques. The following definitions allow capturing these dynamics aspects of the task allocation based on a BDT.

Definition 5. The *pos* is a position function defined from a set of vertexes V of a tree T to the set of natural numbers N as $pos : V \to N$, such that $pos(v) = j$, where j is the numeric position of vertex v with respect to all the vertexes in the same level and counting from left to right.

Definition 6. The *depth* is a function defined from a set of vertexes V of a tree T to the set of natural numbers N as $depth : V \to N$, such that $depth(v) =$ #edges from the root node to v.

Notice that, in **Figure 2**, the identification of a vertex v is given by $v_{i,j}$ where $i = depth(v)$ and $j = pos(v)$.

Definition 7. The w is a weight function defined from the set of edges E of a tree T to the set containing 0 and 1 as $w : E \to \{0, 1\}$, such that $w(e) = 0$ if e connects a parent vertex to its left child, and 1 if it connects to its right child. Let consider a vertex v of a tree T denoted as $v_{i,j}$ such that $i = depth(v)$ and $j = pos(v)$. Then, for $v_{i,x}$ parent of left child $v_{j,y}$ and right child $v_{j,z}$, with their respective edges $e_{i,j,y}$ and $e_{i,j,z}$, with $y < z$, by definition of *depth*, then $w(e_{i,j,y}) = 0$ and $w(e_{i,j,z}) = 1$.

Notice that, the weight function w provides a zero for an edge connecting a father with its left child and a one for the edge connecting it with the right child. In the modeling methodology

http://dx.doi.org/10.5772/67950

here introduced, it is assumed that there exists a data structure called *Task Array*, which may have a reference to a task, if it is initially inserted, and the task has not been taken yet. This *Task Array* is mapped to the location of the leaves of a tree of height H from left to right, i.e., the array has a capacity for referencing a maximum of The quaternion semi − widel tasks. **Figure 1** shows a tree of height 3, with 8 leaves that are associated with the location in the *Task Array*, where references to task are $T0$, $T1\ldots$, $T7$. Accordingly, the following functions are defined.

Definition 8. The *data* function is defined from the set of vertexes V of a tree T to the set of natural numbers N as $data : V \rightarrow N$, where $data(v) =$ # of leaves in the subtree of root v with a task ready to be taken.

Definition 9. The *parent* is a function defined from the set of vertexes of a tree T to itself as $parent : V \rightarrow V$, such that $parent(v) = y$, if v is child of y.

Definition 10. The *val* is a recursive function defined from the set of vertexes V of a tree T of height H, to the set of natural numbers N as $val : V \rightarrow N$, such that:

- (Base): $val(v_{0,0}) = 0$.
- (Induction): $val(v_{i,j}) = val\left(parent(v_{i,j})\right) + w\left(e_{pos(parent(v_{i,j}),\, depth(v_{i,j})),\, j}\right) * 2^{H-i}$, for every pair (i, j), where $1 \leq i \leq H$, $0 \leq j < 2^i$.

In **Figure 2**, for example, $val(v)$ is shown in the left of every vertex v as $val(v)/$, e.g., $val(v_{2,3}) = val\left(parent(v_{2,3})\right) + w\left(e_{pos(parent(v_{2,3}),\, depth(v_{2,3})),\, 3}\right) * 2^{3-2} = val(v_{1,1}) + w\left(e_{pos(v_{1,1}),\, 2,\, 3}\right) * 2^1$ $= val\left(parent(v_{1,1})\right) + w\left(e_{pos(parent(v_{1,1}),\, depth(v_{1,1})),\, 1}\right) * 2^{3-1} + w\left(e_{pos(v_{1,1}),\, 2,\, 3}\right) * 2^1 = val(v_{0,0}) + = 0$ $+ w(e_{0,1,1}) * 4 + w(e_{1,2,3}) * 2 = 0 + 1 * 4 + 1 * 2 = 6$.

Observe that function *val* generates the value of a node based on the value of his parent. Notice that, there are 8 leaves in level 3, and then, every node value is exactly one entry in the *Task Array*. For example, the value 4/ at $v_{3,5}$ points to the entry $T4$. Now, consider the nodes in level 2, which are 4, exactly the half of those at level 3. Every vertex at this level, points to the first entry in a range of 2, i.e., $v_{2,0}$ has a value of 0, $v_{2,1}$ and has a value of 2. Thus, every vertex at level 1, points to the first entry of the two partitions of the *Task Array*, and therefore, $v_{0,0}$ points to the first entry of the whole of the *Task Array*.

For illustration purposes, suppose that there is one node with two threads and eight tasks to be executed. Accordingly, $d = 2$ and $nT = 8$. The CPN System $(\mathcal{N}, \mathbf{M}_0)$ for modeling this problem is shown in **Figure 4**.

Notice that accordingly to the proposed method, a full binary tree is obtained when $nT = 2^n$ for some n, since, in this case, $H = \log_2 nT$ is exactly equal to $\log_2 nT$. In the particular example, $H = 3$, since $nT = 2^3 = 8$. Then, $|P_O| = 2(H + 1) = 8$, with $P_O = \{p_1, p_2, \ldots, p_8\}$; $|P_R| = H = 3$, with $P_R = \{p_9, p_{10}, p_{11}\}$; $|P_A| = 1$ with $P_A = \{p_{12}\}$; $|T| = |P_O| = 8$, with $T = \{t_1, t_2, \ldots, t_8\}$; $o = \{o_1, o_2\}$ since $d = 2$; $r = \{r_1, r_2, \ldots, r_{14}\}$, since $2^{H+1} - 2 = 2^3 - 2 = 14$, and $a = \{a_1, a_2, \ldots, a_8\}$.

The following functions complement the CPN model:

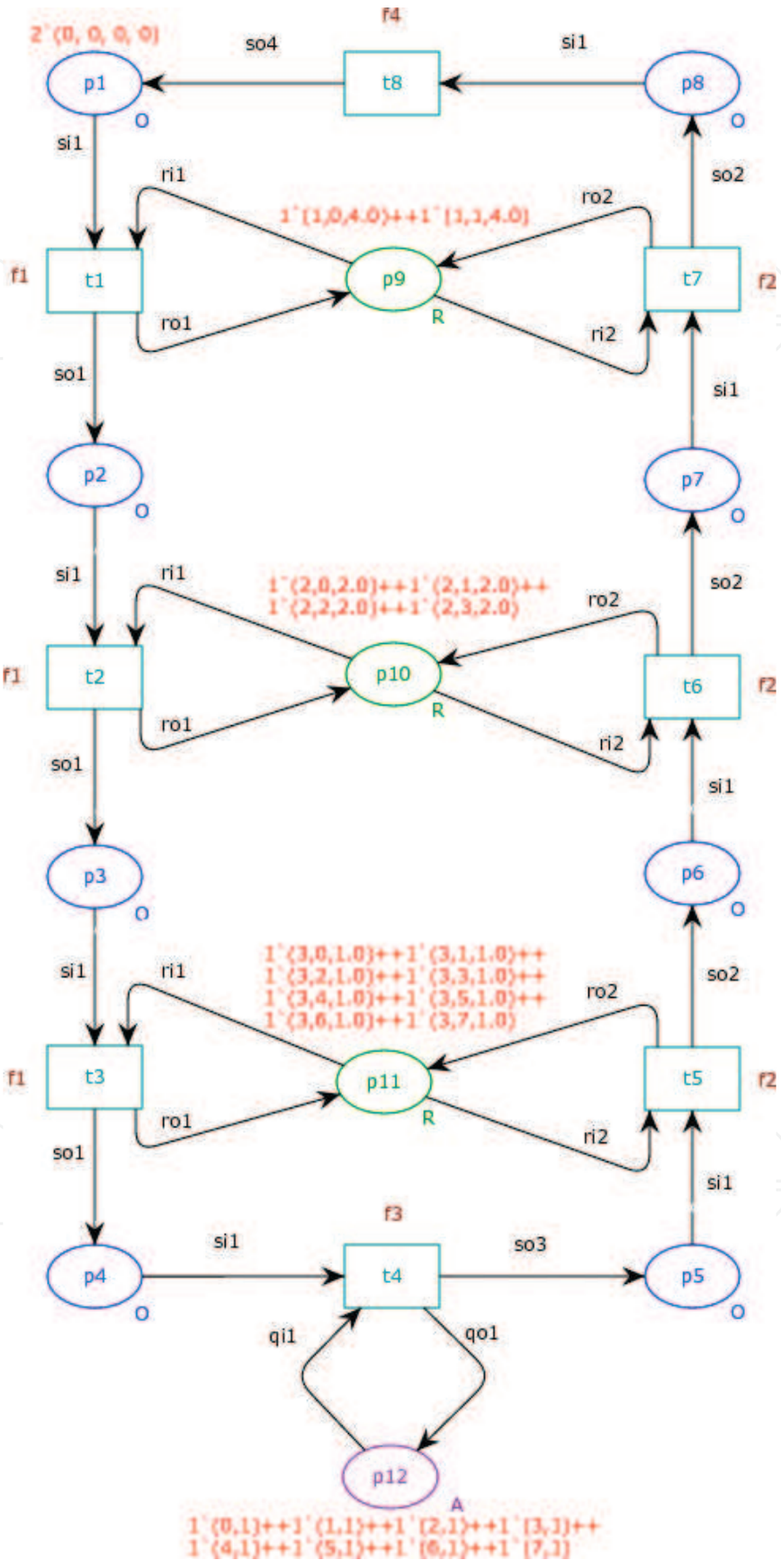

Figure 4. DS CPN with $nT = 8$.

http://dx.doi.org/10.5772/67950

- $o_i = \langle depth_i, pos_i, val_i, success_i \rangle = \langle 0, 0, 0, 0 \rangle$, for $i \in \{1, 2\}$. This marking represents the current state of every process, i.e., at the beginning, processes are at root node $v_{0,0}$ with a value 0. *Success* being 0 means that no task has been assigned to the current process.
- $r_j = \langle depth_j, pos_j, data_j \rangle$, such that $j \in [1, |r|]$, where: $r_1 = \langle 1, 0, 4 \rangle$, $r_2 = \langle 1, 1, 4 \rangle$, $r_3 = \langle 2, 0, 2 \rangle$, $r_4 = \langle 2, 1, 2 \rangle$, $r_5 = \langle 2, 2, 2 \rangle$, $r_6 = \langle 2, 3, 2 \rangle$, $r_7 = \langle 3, 0, 1 \rangle$, $r_8 = \langle 3, 1, 1 \rangle$, $r_9 = \langle 3, 2, 1 \rangle$, $r_{10} = \langle 3, 3, 1 \rangle$, $r_{11} = \langle 3, 4, 1 \rangle$, $r_{12} = \langle 3, 5, 1 \rangle$, $r_{13} = \langle 3, 6, 1 \rangle$, $r_{14} = \langle 3, 7, 1 \rangle$. These markings provide the information about the *data* available for every subtree constructed from the node identified as $v_{depth, pos}$.
- $a_l = \langle pos_l, ty_l \rangle$, such that $l \in [1, nT]$, where: $a_1 = \langle 1, 1 \rangle$, $a_2 = \langle 2, 1 \rangle$, $a_3 = \langle 3, 1 \rangle$, $a_4 = \langle 4, 1 \rangle$, $a_5 = \langle 5, 1 \rangle$, $a_6 = \langle 6, 1 \rangle$, $a_7 = \langle 7, 1 \rangle$, $a_8 = \langle 8, 1 \rangle$. These markings provide the information about the availability of the task located at *pos*, i.e., $ty = 1$ if the task is available, 0 otherwise.

The color mapping is $C_p(p_k) = o$ for $k \in [1, 8]$, $C_p(p_k) = r$ for $k \in [9, 11]$ and $C_p(p_{12}) = a$. That is, the places $p_1, \ldots, p_8$ accept tokens of type O, the places p_9, p_{10}, p_{11} accept tokens of type R, and p_{12} accept tokens of type A. The conditional color mapping is $C_t(t_k) = f_1$ for $k \in [1, 3]$, $C_t(t_k) = f_2$ for $k \in [5, 7]$, and $C_t(t_k) = f_3$ for $k = 4$ and $k = 9$. The *Pre*- and *Post* matrices for the net of this example are shown in **Figure 5**.

Thus, **Figure 4** represents a CPN that captures the structure of a BDT and the behavior of the working threads over the tree of height $H = \log_2 nT = \log_2 8 = 3$. They represents the places and transitions for the traveling from the root down to a leaf, and the reverse way from the leaf up to the root, on the left and right side of the CPN, respectively. Additionally, the central places, marked as "R," represent the "resources" in the system, i.e., the shared memory registers and the tasks.

Consider an initial token in place $p_1 = \langle 0, 0, 0, 0 \rangle$ as described by the initial marking M_0. Now, the binding for t_1 requires two tokens from p_9, besides one initial token in p_1, subject to

Pre

	t_1: f_1	t_2: f_1	t_3: f_1	t_4: f_3	t_5: f_2	t_6: f_2	t_7: f_2	t_8: f_4
p_1:{o_1,o_2}	si_1							
p_2:{o_1,o_2}		si_1						
p_3:{o_1,o_2}			si_1					
p_4:{o_1,o_2}				si_1				
p_5:{o_1,o_2}					si_1			
p_6:{o_1,o_2}						si_1		
p_7:{o_1,o_2}							si_1	
p_8:{o_1,o_2}								si_1
p_9:{r_1,r_2}	ri_1						ri_2	
p_{10}:{r_3,…,r_6}		ri_1				ri_2		
p_{11}:{r_7,…,r_{14}}			ri_1		ri_2			

Post

	t_1: f_1	t_2: f_1	t_3: f_1	t_4: f_3	t_5: f_2	t_6: f_2	t_7: f_2	t_8: f_4
p_1:{o_1,o_2}								so_4
p_2:{o_1,o_2}	so_1							
p_3:{o_1,o_2}		so_1						
p_4:{o_1,o_2}			so_1					
p_5:{o_1,o_2}				so_3				
p_6:{o_1,o_2}					so_2			
p_7:{o_1,o_2}						so_2		
p_8:{o_1,o_2}							so_2	
p_9:{r_1,r_2}	re_1						re_2	
p_{10}:{r_1,…,r_4}		re_1				re_2		
p_{11}:{r_1,…,r_8}			re_1		re_2			

Figure 5. Pre and postmatrices for CPN DS $nT = 8$.

conditions given by f_1: $(o_i, r_j, r_k)|depth_i + 1 = depth_j = depth_k \wedge pos_i * 2 = pos_j \wedge pos_i * 2 + 1 = pos_k$. Since in this case $o_i = o_1 = \langle pos_1 = 0, depht_1 = 0,\ val_1 = 0, success_1 = 0\rangle$ due to si_1, then the required tokens from p_9 are $r_j = r_1 = \langle depth_1 = 0, pos_1 = 1, data_1 = 2^{H-1} = 4\rangle$ and $r_k = r_2 = \langle depth_2 = 1, pos_2 = 1, data_2 = 2^{3-1} = 4\rangle$, where $data_1 = data_2 = 4$, assuming that the Task Array is full of tasks to be attended. Thus, the output of t_1 due to so_1 is $o_1 = \langle depth_1 = depth_1 + 1,\ pos_1 = pos_1 * 2 + h = 1,\ val_1 + h * 2^2 = 4, success_1 = 0\rangle = \langle 1, 1, 4, 0\rangle$, assuming $h = 1$.

The sketches of CPN in **Figure 6** show the three main marking evolutions in the CPN subject to the binding functions f_1, f_2, f_3. The marking evolution subject to f_1 from the current state is shown in (a). In (b), the tokens o_i, r_j, r_k are taken by the depicted transition, since f_1: $(o_i, r_j, r_k)|depth_i + 1 = depth_j = depth_k \wedge pos_i*2 = pos_j \wedge pos_i*2 + 1 = pos_k$, then $o_i =< 1, 1, 0, 0 >$, $r_j =< 2, 2, 2 >$, $r_k =< 2, 3, 2 >$. In (c), the transition has fired, then $o_1 =< 2, 3, 6, 0 >$, since $so_1(\langle depth_i, pos_i, val_i, success_i\rangle) = \langle depth_i + 1, pos_i * 2 + h_i, val_i + (h_i) * 2^{H-depth_i+1}, success_i\rangle$ assuming that $h_i = 1$. The tokens r_j, r_k remain the same, since $success = 0$. In (d), the marking evolution subject to f_3 from the current marking $< 3, 6, 6, 0 >$ is updated to the marking $< 3, 6, 6, 1 >$

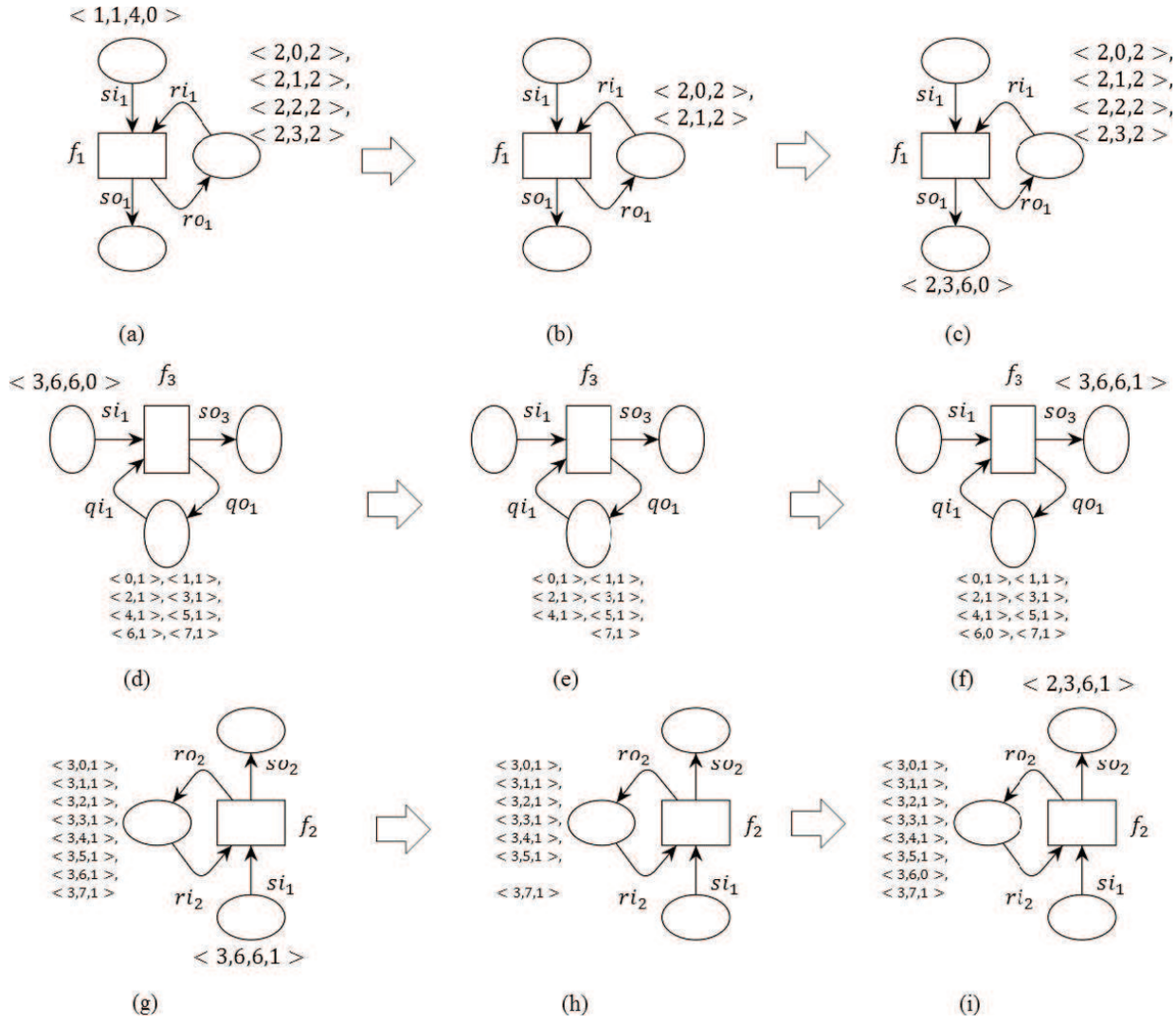

Figure 6. Binding and firing of a CPN transition.

http://dx.doi.org/10.5772/67950

as shown in (f), since the task reference represented by the marking $< 6, 1 >$ is available as shown in (e). Notice that, this marking is updated to $< 6, 0 >$ as shown in (f), as expected. In (g), the input state of the transitions subject to f_2 is represented with the marking $< 3, 6, 6, 1 >$. Then, it is binding by f_2 with the token $< 3, 6, 1 >$, as illustrated in (h). Thus, the input token $< 3, 6, 6, 1 >$ is updated to $< 2, 3, 6, 1 >$ by so_2, as well as, $< 3, 6, 1 >$ is updated to $< 3, 6, 0 >$ by ro_2, as shown in the section (i) of the figure.

One of the main advantages of modeling the task allocation problem by using CPN's is the possibility of varying the parameters during the simulation process, as well as the flexibility of the net structure in order to cope with a greater number of task to be attended. One of the key parameters for the simulation of the problem is the number of tokens of type O, which represents the number of processes or threads in a specific problem. The other important parameter is the decision function α, which is related to the spreading of processes or threads through the tree, by updating of *depth* and *pos* functions. It is clear, for example, that the distribution of the processes or threads at every level in the tree structure directly influences the contention at the acquisition of the tasks.

The next section explains the simulation process that is possible to execute with the proposed modeling methodology and how these simulations can help explore different parameters in order to optimize the task allocation problem.

5. Simulation of the CPN model

This section shows how to simulate a CPN model as that obtained by the methodology detailed in the previous subsection. The simulation process allows to investigate the performance of the model under parametric variations. First, the general characteristics of the simulation framework are introduced. Then, the simulation and results of a CPN model, representing a problem with 1024 tasks, are given.

In order to simplify the construction of the CPN model by using the herein proposed methodology, and thus, simulate its behavior, a CPN tool is used [43]. This environment allows editing, simulating and analyzing CPN models [36–42] and was successfully used for a variety of applications areas [44–48].

The model parameters that have to be considered are a number of initially available tasks $(nTasks)$, the height of the tree with respect to the number of tasks (H), and the decision function (α). For this example, those values are $nTasks = 1024$, $H = \log_2 1024 = 10$, and $\alpha = (r < data_j/(data_j + data_k))$ for a given o_i, r_j, r_k such that $depth_i + 1 = depth_j = depth_k \wedge pos_i * 2 = pos_j \wedge pos_i * 2 + 1 = pos_k$ and where r is a randomly generated number. These parameters are fixed during every simulation process. However, the number of threads is varying between simulations, with $d = 2^n,\ 1 \leq n \leq nTasks$, for a total of 10 simulations runs.

In the simulation framework, there are different measurements of parameters that can be obtained. For illustrative purposes, in this example, the attention is focused on study the impact on the performance of the task allocation procedure when the number of processes, or working threads, is increasing.

Table 1 summarizes some results obtained from the simulations performed on the *CPN* tools framework for a model with model described previously. The number of tasks was constant, while the number of processes was increased by a power of two, represented in the first column. The second column represents the average of events required per process to complete all the tasks. It provides a measure of the speed by increasing the number of processes. The third column represents the total number of readings to the task array. The forth column represents the total number of read collisions, i.e., when a process reads a task that was previously attended by other process. The fifth column represents the average of the reads to the task array executed by each process. The sixth column shows the average of the failed reads or collisions executed by each process. The seventh column shows the proficiency of the execution by the total amount of processes. Finally, the eighth column represents the average of total reads, including the failed ones, to the task array required per task.

The plot in **Figure 7** shows the average of event, in the simulation framework, that each process required for completing all the tasks in the array. Notice that, as the number of processes increases exponentially, the number of events per processes decreases almost logarithmically.

The plot in **Figure 8** shows the average of reads that each process has performed to the task array for the completion of all the tasks. As in the previous figure, notice that the number of reads decreases almost logarithmically as the number of processes increases exponentially.

The plot in **Figure 9** shows the average of collisions, i.e., a process's read of a task that was previously attended by other process, as the number of processes increases. The figure shows that the maximum number of average collisions occurs when the number of processes is 32 in these experiments. The collisions, or failed reads, are highly influenced by the decision rule α.

The plot in **Figure 10** shows the proficiency of attending all the tasks in the array as the number of processes increases. The proficiency increases almost exponentially as the number of

Processes (d)	Average events	Reads	Collisions	Reads per process	Fails per process	Proficiency	Read per task
1	22528.00	1024.00	0.00	1024.00	0.00	1.00	1.00
2	11293.25	1027.00	3.00	513.50	1.50	1.99	1.00
4	5667.00	1030.67	6.67	257.67	1.67	3.97	1.01
8	2853.02	1038.00	14.00	129.75	1.75	7.89	1.01
16	1443.81	1051.33	27.33	65.71	1.71	15.58	1.03
32	743.34	1081.67	57.67	33.80	1.80	30.29	1.06
64	384.67	1117.33	93.33	17.46	1.46	58.65	1.09
128	202.77	1175.00	151.00	9.18	1.18	111.55	1.15
256	108.85	1265.67	241.67	4.94	0.94	207.12	1.24
512	55.39	1271.00	247.00	2.48	0.48	412.50	1.24
1024	27.70	1285.50	261.50	1.26	0.26	815.70	1.26

Table 1. Simulation results for $nTasks = 1024$.

http://dx.doi.org/10.5772/67950

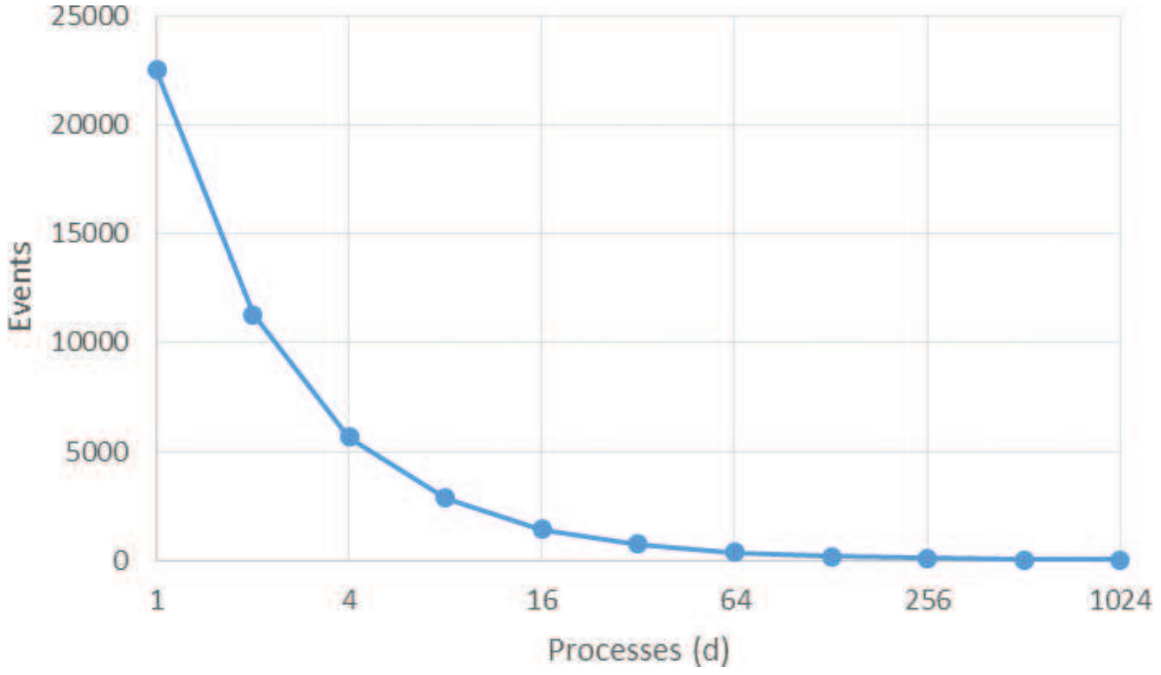

Figure 7. Average of events per process.

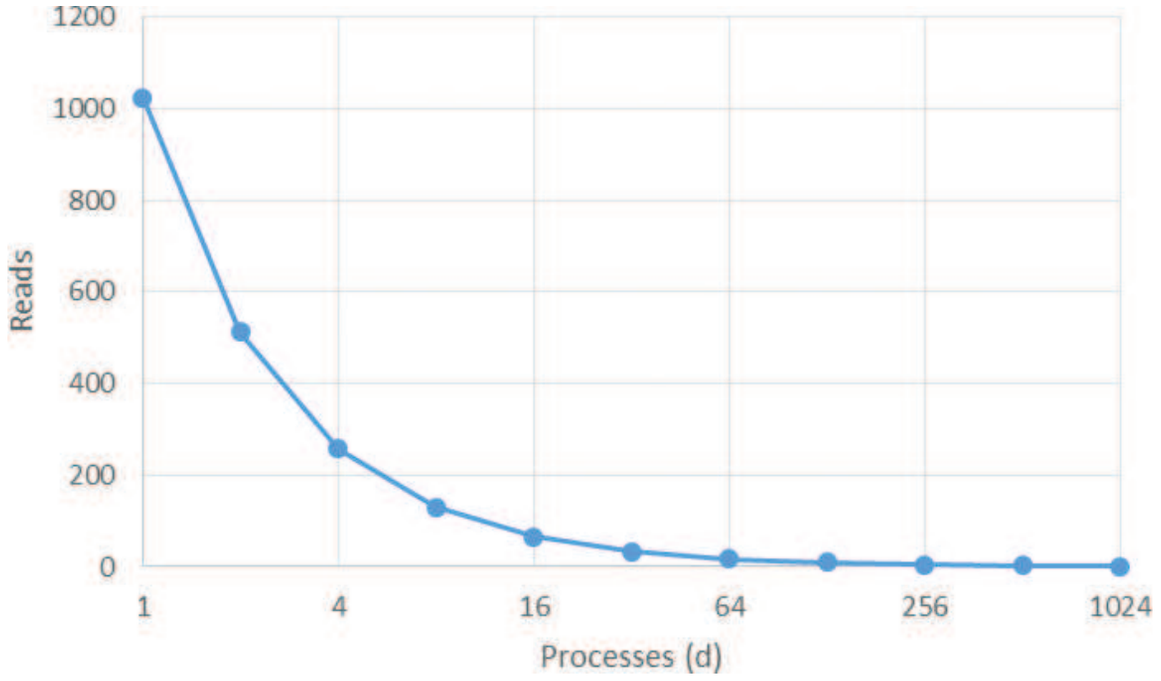

Figure 8. Average of reads per process.

processes increases to 256. The ideal behavior is to double the proficiency as the number of processes also double. However, the collisions at reading the tasks undermine this proficiency, as expected. Notice that, the plot is logarithmic.

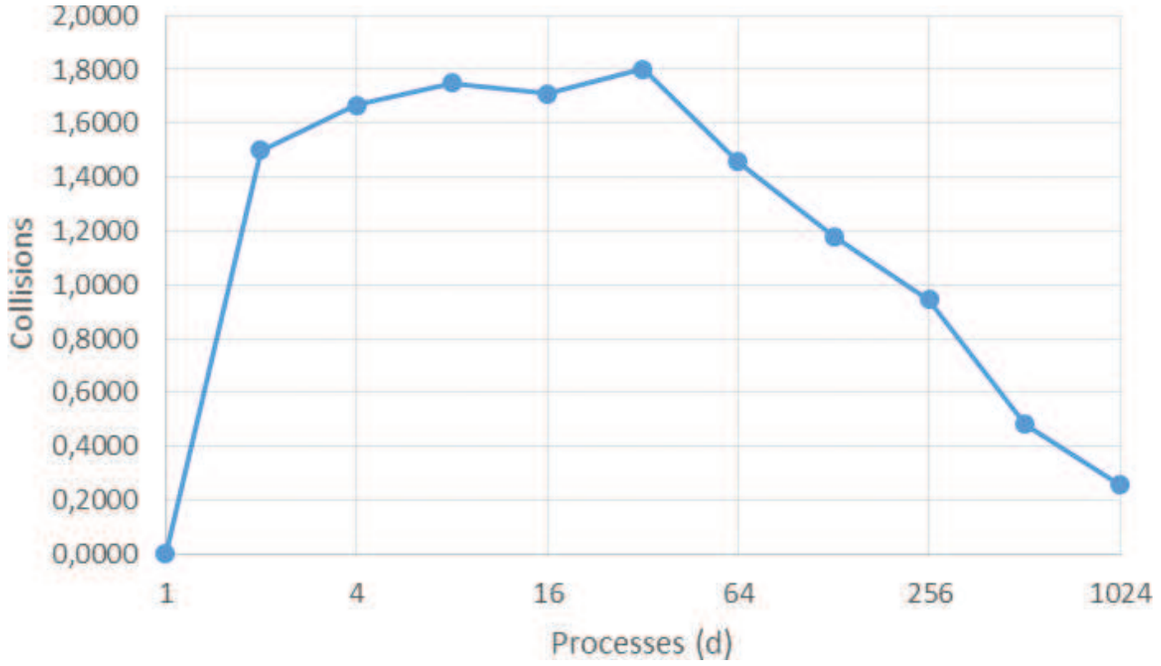

Figure 9. Average of collisions per process.

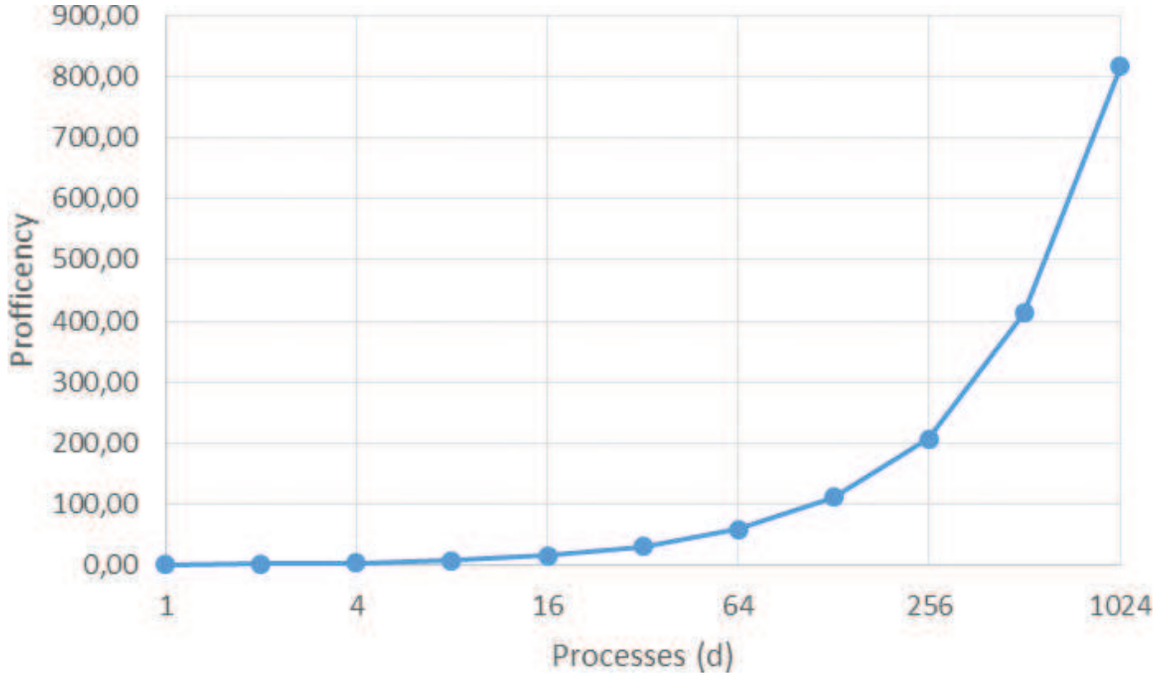

Figure 10. Average of proficiency.

The plot in **Figure 11** shows the average of work that each process has to do for completing a task, measured as the number of reads. This number slightly increases as the number of processes increases due to the growth in the number of collisions.

http://dx.doi.org/10.5772/67950

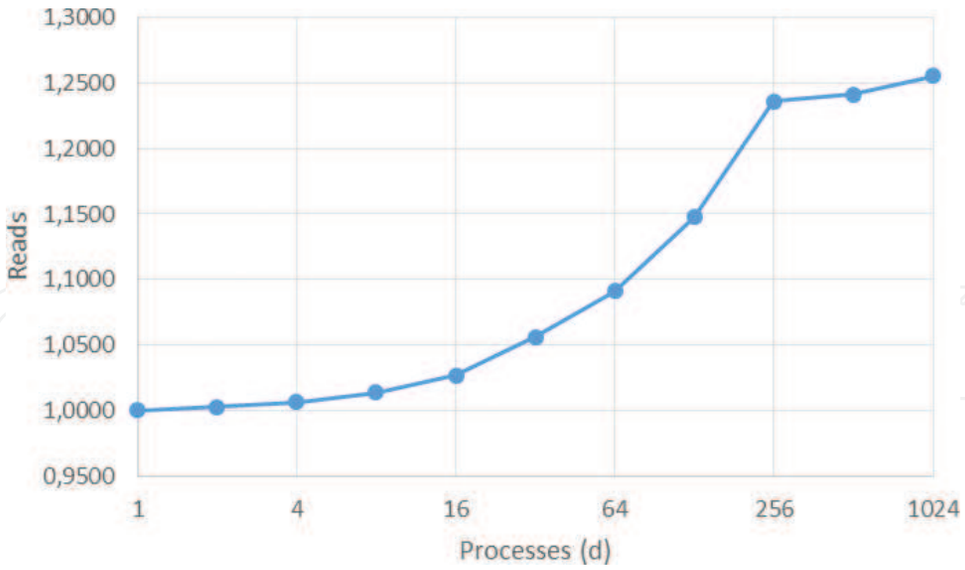

Figure 11. Average of reads per task.

6. Conclusions

This work presented a framework, based on Colored Petri nets, for the modeling and simulation of task allocation problems, which arises in environments such as grid or cluster of computers. The framework allows the construction of complex problems in a compact way. Additionally, a simulation process of the constructed models permits the study of different key aspects of the allocation strategies. Some analysis that could be performed includes the impact of different decision rules of the processes for allocating the tasks, the effect of a greater number of processes in the contention of the task's acquisition or the ration of the increased speed to the attention of tasks by a greater number of processes, among others. Additionally, the methodology allows with ease the construction of structures for an incremental model construction and its respective simulation process.

The proposed methodology allows with ease the extension to $n - ary$ tree structures as well as tree structures with a greater number of tasks and their respective simulation process.

Author details

Mildreth Alcaraz-Mejia*, Raul Campos-Rodriguez and Marco Caballero-Gutierrez

*Address all correspondence to: mildreth@iteso.mx

Electronics, Systems and Informatics Department, ITESO University, Tlaquepaque, Jalisco, Mexico

References

[1] Lee, E. Y. S., and Tsuchiya, M., A task allocation model for distributed computing systems. IEEE Transactions on Computers, 1982, vol 100, no 1, pp. 41–47.

[2] Han, Y., Jiang, C., and Luo, X., Resource scheduling model for grid computing based on sharing synthesis of Petri net, in Proceedings of the 9th International Conference on Computer Supported Cooperative Work in Design, Coventry, UK, 2005, pp. 367–372.

[3] Han, Y., Jiang, C., and Luo, X., Modelling and performance analysis of grid task scheduling based on composition and reduction of Petri nets, in Proceedings of the 5th International Conference on Grid and Cooperative Computing, Changsha, China, 2006, pp. 331–334.

[4] Zhao, X., Wang, B., and Xu, L., Grid application scheduling model based on Petri net with changeable structure, in Proceeding of 6th International Conference on Grid and Cooperative Computing, Los Alamitos, CA, 2007, pp. 733–736.

[5] Han, Y., Jiang, C., and Luo, X., Resource scheduling scheme for grid computing and its Petri net model and analysis, Parallel and Distributed Processing and Applications, Lecture Notes in Computer Science, vol. 3759, G. Chen et al., editors, Heidelberg: Springer, pp. 530–539, 2005.

[6] Hu, Z. G., Hu, R., Gui, W. H., Chen, J. E., and Chen, S. Q., General scheduling framework in computational grid based on Petri net, Journal of Central South University of Technology, vol. 12, no. 1, pp. 232–237, 2005.

[7] Shojafar, M., Barzegar, S., and Meybodi, M. R., A new method on resource scheduling in grid systems based on hierarchical stochastic Petri net, in Proceedings of the 3rd International Conference on Computer and Electrical Engineering, Chengdu, China, 2010, pp. 175–180.

[8] Buyya, R., and Murshed, M., GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience, vol. 14, no. 13–15, pp. 1175–1220, 2002.

[9] Zhovtobryukh, D., A Petri net-based approach for automated goal-driven web service composition, Simulation, January 2007; vol. 83, 1: pp. 33–63.

[10] Haggarty, O. J., Knottenbelt, W. J., and Bradley, J. T., Distributed response time analysis of GSPN models with MapReduce, Simulation, August 2009; vol. 85, 8: pp. 497–509.

[11] Narciso, M., Piera, M. A., and Guasch, A., A methodology for solving logistic optimization problems through simulation, Simulation, May/June 2010; vol. 86, 5–6: pp. 369–389.

[12] Xiong, Z. G., Zhai, Z. L., Zhang, X. M., and Xia, X. W., Grid workflow service composition based on colored petri net. JDCTA: International Journal of Digital Content Technology and its Applications, 2011, vol. 5, no 5, pp. 125–131.

[13] Camilli, M., Petri nets state space analysis in the cloud. In 2012 34th International Conference on Software Engineering (ICSE). pp. 1638–1640.

[14] Longo, F., Ghosh, R., Naik, V. K., and Trivedi, K. S., A scalable availability model for infrastructure-as-a-service cloud. In 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN), IEEE 2011. pp. 335–346.

[15] Wei, B., Lin, C., and Kong, X., Dependability modeling and analysis for the virtual data center of cloud computing. In 2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC), IEEE, 2011. pp. 784–789.

[16] Li, X., Fan, Y., Sheng, Q. Z., Maamar, Z., and Zhu, H., A petri net approach to analyzing behavioral compatibility and similarity of web services. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 2011, vol. 41, no 3, pp. 510–521.

[17] Muñoz, D. M., Correcher, A., García, E., and Morant, F., Stochastic DES fault diagnosis with coloured interpreted Petri nets. Mathematical Problems in Engineering, vol. 2015, Article ID 303107, 13 pages, 2015. doi:10.1155/2015/303107.

[18] Ghainani, A. T., Mohd Zin, A. A., and Ismail, N. A. M., Fuzzy timing Petri net for fault diagnosis in power system. Mathematical Problems in Engineering, vol. 2012, Article ID 717195, 12 pages, 2012. doi:10.1155/2012/717195.

[19] Alcaraz-Mejia, M., Lopez-Mellado, E., Ramirez-Treviño, A., Rivera-Rangel, I., Petri net based fault diagnosis of discrete event systems, In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. pp. 4730–4735, October 2003.

[20] Latorre, J. I., Jiménez, E., and Pérez, M., The optimization problem based on alternatives aggregation Petri nets as models for industrial discrete event systems, Simulation, March 2013; vol. 89, no 3: pp. 346–361.

[21] Latorre, J. I., and Jiménez, E., Simulation-based optimization of discrete event systems with alternative structural configurations using distributed computation and the Petri net paradigm, Simulation, November 2013; vol. 89, no 11: pp. 1310–1334.

[22] Alcaraz-Mejia, M., and Lopez-Mellado, E., Petri net model reconfiguration of discrete manufacturing systems. In Alexandre Dolgui Gerard Morel Carlos Pereira Editors: Information Control Problems in Manufacturing. 1st edition, Editorial Elsevier Science, 2006. pp. 517–552. eBook ISBN: 9780080478487, ISBN: 9780080446547, Page Count: 2480.

[23] Alcaraz-Mejia, M., and Lopez-Mellado, E., Fault recovery of manufacturing systems based on controller reconfiguration. In 2006 IEEE/SMC International Conference on System of Systems Engineering, IEEE, 2006. p. 6.

[24] Alcaraz-Mejia, M., Lopez-Mellado, E., and Ramirez-Treviño, A., A redundancy based method for Petri net model reconfiguration. In IEEE International Conference on Systems, Man and Cybernetics, 2007. IEEE, 2007. pp. 1382–1387.

[25] Chen, S. J., Zhan, T. S., Huang, C. H., Chen, J. L., and Lin, C. H. (2015). Nontechnical loss and outage detection using fractional-order self-synchronization error-based fuzzy petri nets in micro-distribution systems. IEEE Transactions on smart grid, 6(1), 411–420.

[26] Muñoz, D. M., Correcher, A., García, E., and Morant, F., Identification of stochastic timed discrete event systems with st-IPN. Mathematical Problems in Engineering, vol. 2014, Article ID 835312, 21 pages, 2014. doi:10.1155/2014/835312.

[27] Alistarh, D., Bender, M., Gilbert, S., and Guerraoui, R., How to allocate tasks asynchronously. 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012, pp. 331–340, 2012.

[28] Jevtić, A., Gutiérrez, A., Andina, D., and Jamshidi, M., Distributed bees algorithm for task allocation in swarm of robots. IEEE Systems Journal, 2012, vol. 6, no 2, pp. 296–304.

[29] Delle Fave, F. M., Rogers, A., Xu, Z., Sukkarieh, S., and Jennings, N. R., Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. In 2012 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2012, pp. 469–476.

[30] Johnson, L., Choi, H. L., Ponda, S., and How, J. P., Allowing non-submodular score functions in distributed task allocation. In 2012 IEEE 51st Annual Conference on Decision and Control (CDC). IEEE, 2012, pp. 4702–4708.

[31] Zhao-Pin, S. U., Jiang, J. G., Liang, C. Y., and Zhang, G. F., A distributed algorithm for parallel multi-task allocation based on profit sharing learning. Acta Automatica Sinica, 2011, vol. 37, no 7, pp. 865–872.

[32] Macarthur, K. S., Stranders, R., Ramchurn, S. D., and Jennings, N. R., A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In Proceedings of the 25th Association of the Advancement on Artificial Intelligence Conference (AAAI 2011), pp. 356–362, 2011. San Francisco, USA.

[33] Alistarh, D., Aspnes, J., Bender, M. A., Gelashvili, R., and Gilbert, S. Dynamic task allocation in asynchronous shared memory. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 2014. pp. 416–435.

[34] Diestel, R., Graph Theory. Springer-Verlag, Heidelberg. Berlin. Graduate Texts in Mathematics, vol. 173, 3rd edition, 2005.

[35] Girault, C. and Valk, R., Petri Nets for Systems Engineering: A Guide to Modelling, Verification, and Applications, July 30, 2001, Springer-Verlag Berlin Heidelberg, Newyork, 2003. ISBN: 3642074472 9783642074479. Series ISSN: 1431–2654. vol. 2, no 1.

[36] Jensen, K., Coloured Petri nets: basic concepts, analysis methods and practical use. Springer Science and Business Media, 2013.

[37] Jensen, K., Kristensen, L. M., and Wells, L., Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer, 2007, vol. 9, no 3–4, pp. 213–254.

[38] Ratzer A.V. et al. (2003) CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In: van der Aalst W.M.P., Best E. (eds) Applications and Theory of Petri Nets 2003. ICATPN 2003. Lecture Notes in Computer Science, vol. 2679. Springer, Berlin, Heidelberg.

[39] Wells, L., Performance analysis using CPN tools. In Proceedings of the 1st international conference on Performance evaluation methodologies and tools. ACM, 2006. p. 59.

[40] Westergaard, M., CPN Tools 4: Multi-formalism and extensibility. In Application and Theory of Petri Nets and Concurrency. Springer Berlin Heidelberg, 2013. pp. 400–409.

[41] Westergaard, M. and Slaats, T., CPN Tools 4: a process modeling tool combining declarative and imperative paradigms. Automatic Control and Computer Sciences, 2013, vol. 47, no 7, pp. 393–402.

[42] Dworzański, L. W., and Lomazova, Irina A., CPN Tools-assisted simulation and verification of nested Petri nets. Automatic Control and Computer Sciences, 2013, vol. 47, no 7, pp. 393–402.

[43] Cpntools.org, (2015). CPN Tools Homepage. [online] Available at: http://cpntools.org/ [Accessed 24 Jul. 2015].

[44] Machado, R. J., Lassen, K. B., Oliveira, S., Couto, M., and Pinto, P., Requirements validation: execution of UML models with CPN tools. International Journal on Software Tools for Technology Transfer, 2007, vol. 9, no 3–4, pp. 353–369.

[45] Vanderfeesten, I., van der Aalst, W., and Reijers, H. A., Modelling a product based workflow system in CPN tools. In Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005). 2005. pp. 99–118.

[46] Störrle, H., Semantics and verification of data flow in UML 2.0 activities. Electronic Notes in Theoretical Computer Science, 2005, vol. 127, no 4, pp. 35–52.

[47] Yi, X., and Kochut, K. J., A cp-nets-based design and verification framework for web services composition. In Web Services, 2004. Proceedings. IEEE International Conference on. IEEE, 2004. pp. 756–760.

[48] Rozinat, A., Wynn, M. T., van der Aalst, W. M., terHofstede, A. H., and Fidge, C. J., Workflow simulation for operational decision support. Data and Knowledge Engineering, 2009, vol. 68, no 9, pp. 834–850.

IntechOpen