An analytical model to evaluate the performance of cluster architectures

Leonardo Brenner — CAPES-CPAD-PUCRS,Brazil — *lbrenner@inf.pucrs.br*
César A. F. De Rose — CPAD-PUCRS,Brazil — *derose@inf.pucrs.br*
Paulo Fernandes — CNPq-PUCRS,Brazil — *paulof@inf.pucrs.br*

Abstract

*This paper presents a rather simple method to model cluster architectures using Stochastic Automata Networks (*SAN*) formalism. Although, the modeling power of* SAN*, the presented models are quite simple and do not fully describe the complexity of existing clusters. It is not the purpose of this paper to proposed ready-to-evaluate models to real clusters, but just introduce the formalism to the parallel architectures community. The conclusion of this paper discusses the further steps in order to perform a real case evaluation.*

1. Introduction

Cluster architectures are becoming a very attractive alternative when high performance is needed. With a very good cost/performance relation and good scalability cluster systems are becoming more popular in universities, research labs and industries. Because clusters are usually build with of the shelf components they are highly configurable. Therefore, users may choose among several node configurations and interconnection networks. These choices will have a direct impact on global system performance and small system peculiarities, like the node cache size or how the nodes are connected to the network, and it may be sufficient to shift bottlenecks inside similar systems [2]. The price of all this flexibility is paid by the programmers. To generate efficient code, parallel programmers need a very good knowledge of the target cluster and the optimizations made for this machine may not be effective in other similar cluster configurations. This paper presents generic models to analytically evaluate the performance of NORMA clusters (No Remote Memory Architecture [6]) using Stochastic Automata Network(SAN).

SAN is a formalism to model complex systems as a collection of interacting subsystems[4]. SAN is particularly useful to model parallel systems because its basic primitives are parallel actions and synchronism among them[8]. Such a model could be used to help system administrators choose the right cluster configuration for their needs or to help programmers to optimize parallel applications.

Although, the modeling power of SAN, the presented models are quite simple and do not fully describe the complexity of real clusters implementations. It is not the purpose of this paper to proposed ready-to-evaluate models to real clusters, but just introduce the formalism to the parallel architectures community.

In order to do so, the next section informally presents the SAN formalism. Section 3 presents the generic cluster models for some possible interconnection architectures. Finally, the conclusion discusses the further steps in order to perform a real case evaluation.

2. Stochastic Automata Networks Formalism

In this section we introduce the basic concepts of the SAN formalism. The reader interested in a formal description of the formalism can consult previous publications [5, 4, 7], or the original work on the subject [8]. However, a previous knowledge about Kronecker products and tensor algebra [3] is helpful to understand the foundations of the formalism.

The SAN formalism describes a complete system as a collection of subsystems that interact with each other. Each subsystem is described as a stochastic automaton, *i.e.*, an automaton in which the transitions are labeled with probabilistic and timing information. Hence, one can build a continuous-time stochastic process[1] related to the SAN. Therefore, any SAN model has an equivalent Markov chain [9].

The state of a SAN model, called *global state*, is defined by the combination of the states of all automata, each of them being called a *local state*. Figure 1 represents a SAN model with two automata completely independent and its equivalent Markov chain.

In this figure (Figure 1), and in the other SAN models of this paper we adopt the following notation:

Let

N the number of automata in a SAN model;

$\mathcal{A}^{(i)}$ the i-th automaton of a SAN model, numbering the first automaton as $\mathcal{A}^{(1)}$ ($i \in 1..N$);

n_i the size (number of states) of automaton $\mathcal{A}^{(i)}$ ($i \in 1..N$);

$j^{(i)}$ the j-th state of the automaton $\mathcal{A}^{(i)}$, numbering the first state of the first automaton as $0^{(1)}$ ($j \in 0..(n_i - 1)$);

l_j a local event identifier (the index j has no particular semantic);

s_j a synchronized event identifier (the index j has no particular semantic);

τ_j a constant rate of given a transition (the index j has no particular semantic);

π_j a constant probability of given a transition (the index j has no particular semantic);

f_j a functional rate of given a transition (the index j has no particular semantic);

g_j a functional probability of given a transition (the index j has no particular semantic);

[1] The time scale of SAN models can be continuous (time is described as transition rate) or discrete (time is described as a transition probability). In the context of this paper only continuous-time SAN will be considered, although discrete-time SAN can also be employed without any loss of generality.

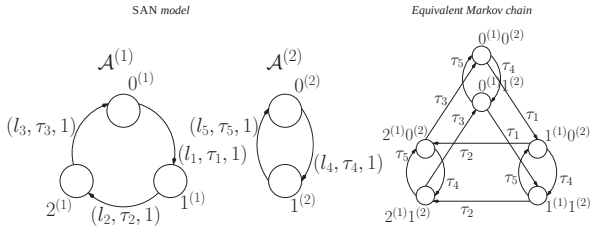


Figure 1. SAN **model with 2 independent automata**

The basics of any SAN model is the concept of events, *i.e.*, a thing that causes the change of the SAN global state. There are two kinds of events:

- *local events*, that change the SAN global state by changing the local state of one single automaton;
- *synchronized events*, that change the SAN global state by changing the local state of two or more automata simultaneously.

Note that in the model of Figure 1, there is no interaction between the two automata. In the following sections we will extend this model to illustrate the use of two SAN primitives (the *synchronized events* and the *functional rates*) to represent the interactions among automata.

The model in Figure 1 has only local events. Local events are represented by the *name* of the event (an identifier[2]), its *rate* of occurrence and a *probability* of occurrence. The probability of occurrence quantifies a choice among all transitions corresponding to a same event that can be fired from the same local state, and therefore cannot be fired simultaneously.

2.1. Synchronized Events

While the local events change the global state passing from a global state to another that differs only by one local state, the synchronized events can change simultaneously more than one local state, *i.e.*, two or more automata can change their local states simultaneously. The model in Figure 1 has only local events, Figure 2 represents a slightly different model where a synchronized event (s_1) has been included.

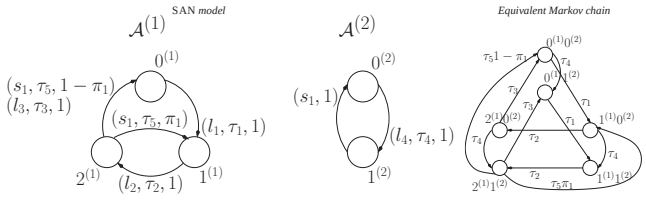


Figure 2. SAN **model with 2 automata and a synchronized event**

The occurrence of a synchronized event *forces* all automata concerned to fire a transition corresponding to this event. Arbitrarily, one of these automata will be chosen as *master* and the other concerned automata will be defined as *slaves*. This choice of a master automaton *causing* the synchronizing event in the slaves automata may not necessarily

[2] In this paper we use an indexed roman letter l as identifier, but to all purposes, any identifier can be used.

correspond to the modeled reality. Other synchronized behaviors, *e.g.*, *rendez-vous*, may be modeled with SAN synchronized events in the same manner with no loss of generality. It is also important to notice that the definition of the master automaton is made to each synchronized event. Therefore, a same automaton can be master of a given event and slave to other events.

The synchronized events in the master automata are represented as in local events, *i.e.*, with the *name* of the synchronized event (an identifier), its *firing rate* and its *probability of occurrence*.

- The *name* of the event is necessary to identify which transitions must fire simultaneously;
- The *firing rate* describes the rate at which the event occurs;
- The *probability of occurrence* quantifies a choice among all transitions corresponding to a synchronizing event that can be fired from the same local state.

In the slave automata, the the transition that may be fired by a synchronized event are represented with just the *name* of the synchronized event and its *probability of occurrence*. There is no need to indicate the rate of the event, since it will be indicate in the master automaton.

In the model of Figure 2, transitions $0^{(1)} \rightarrow 1^{(1)}$, $1^{(1)} \rightarrow 2^{(1)}$, $2^{(1)} \rightarrow 0^{(1)}$, and $0^{(2)} \rightarrow 1^{(2)}$ may be fired due to the local events l_1, l_2, l_3 and l_4 respectively. Transitions $2^{(1)} \rightarrow 0^{(1)}$, $2^{(1)} \rightarrow 1^{(1)}$, and $1^{(2)} \rightarrow 0^{(2)}$ may be fired by the occurrence of the synchronized event s_1. Note that transition $2^{(1)} \rightarrow 0^{(1)}$ can be fired by the occurrence of a local event (l_3), with rate τ_3, or by the occurrence of the synchronized event s_1. Notice that it is not important to give further definitions when this transition will fired according to the local or the synchronized events. In this example, the occurrence of the synchronized event s_1 (which happens at rate τ_5) leads to one of the two situations:

- automata $\mathcal{A}^{(1)}$ goes from state $2^{(1)}$ to state $1^{(1)}$ and at the same time automata $\mathcal{A}^{(1)}$ goes from state $0^{(2)}$ to state $1^{(2)}$ (with probability π_1); or
- automata $\mathcal{A}^{(1)}$ goes from state $2^{(1)}$ to state $0^{(1)}$ and at the same time automata $\mathcal{A}^{(1)}$ goes from state $0^{(2)}$ to state $1^{(2)}$ (with probability $1 - \pi_1$).

2.2. Functional Rates

The use of functional rates is the second form of interaction among automata. A functional rate is no longer a non-negative Real number (like the ordinary rates), but a discrete function of the local state of some automata over the non-negative Real numbers. Figure 3 represents a variation of the example in Figure 2. In this new example, the rate of the local event l_4 is no longer a constant rate, but it is a function called f_1 defined as:

$$f_1 = \begin{cases} \lambda_1 & \text{if automaton } \mathcal{A}^{(1)} \text{ is in the state } 0^{(1)}; \\ 0 & \text{if automaton } \mathcal{A}^{(1)} \text{ is in the state } 1^{(1)}; \\ \lambda_2 & \text{if automaton } \mathcal{A}^{(1)} \text{ is in the state } 2^{(1)}; \end{cases}$$

In this model, the firing of the transition from state $0^{(2)}$ to $1^{(2)}$ occurs with rate λ_1 if automaton $\mathcal{A}^{(1)}$ is in state $0^{(1)}$, or with rate λ_2 if automaton $\mathcal{A}^{(1)}$ is in state $2^{(1)}$. If automaton $\mathcal{A}^{(1)}$ is in state $1^{(1)}$ the transition from state $0^{(2)}$

to $1^{(2)}$ does not occur (rate 0). Using the SAN notation employed by the software tool PEPS2000, the expression of this function is:

$$f = \left[\lambda_1 \left(st(\mathcal{A}^{(1)}) == 0^{(1)}\right)\right] + \left[\lambda_2 \left(st(\mathcal{A}^{(1)}) == 2^{(2)}\right)\right]$$

The interpretation of a function can be viewed as the evaluation of an expression of non-typed programming languages, *e.g.*, C language. Each comparison is evaluated to 1 for true and to 0 for false.

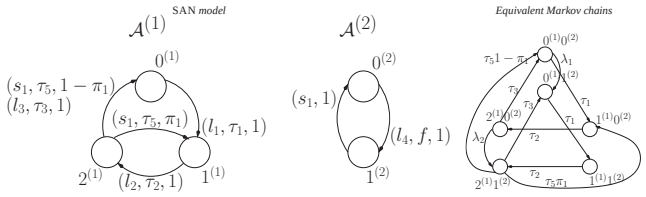


Figure 3. A SAN model with 2 automata, 1 synchronized event, and 1 functional rate

Note that the use of functional rates is not limited to local event rates. In fact, for synchronized events not only the event rate, but also the probability of occurrence, can be expressed as a function. The use of functional transitions is a powerful primitive of the SAN formalism, since it allows to describe very complex behaviors in a very compact format. The computational costs to handle functional rates has decreased significantly with the developments of numerical solutions for SAN models, *e.g.*, the algorithms for generalized tensor products.

2.3. Numerical Results

The solution of a SAN model usually correspond to the computation of the stationary probability distribution of each global state. The integration of such probability vector allows the stationary probability distribution of each local state of each automaton. More sophisticated integrations of the probability vector are possible. The PEPS software tool [10] is an academic software to solve SAN models and to integrate the stationary solution in many different ways. PEPS software tool implements most recent numerical techniques [1, 4] to efficiently solve SAN models. In this paper, the possible quantitative indexes that can be computed to each cluster architecture model will not be discussed. The reader may just imagine the basic stationary probability distribution of the local states. More elaborated results are consequence of this basic values, but their formulation is out of this paper scope.

3. Modeling Clusters with Stochastic Automata Network

In this section we introduce some simple models of clusters machines using SAN. We assume each node working independently and communicating with the others through a protocol of exchanging messages. This model does not take directly into account the manner as each computer executes its tasks, nor the protocol of communication used between them. In such way, each automaton represents the state where the node is and not how it executes the tasks relative to each state. As in any continuous-time model, the transition rates among the states will represent the inverse of time spend in the state.

Three different clusters interconnections are modeled: bus, ring and torus. Each interconnection has particular characteristics and needs to be model in a different way.

3.1. Bus Cluster

The first model describe a cluster with N nodes interconnected by a single bus, *e.g.*, an Ethernet. The communication media (bus) is shared by all nodes, and the bus access is granted to a node at time. Figure 4 illustrate such architecture.

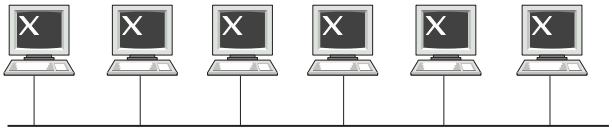
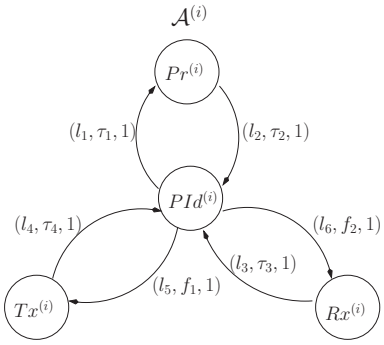


Figure 4. Bus Interconnection Architecture

The SAN model proposed to represent this architecture has N identical automata, each representing the possible states of a node. Figure 5 represents following the four possible states:

- $PId^{(i)}$ - the node i is idle (or waiting);
- $Pr^{(i)}$ - the node i is processing a task;
- $Tx^{(i)}$ - the node i is transmitting a message;
- $Rx^{(i)}$ - the node i is receiving a message;



$$f_1 = \left(nb\ Tx^{(i)} == 0\right) \tau_5 \qquad f_2 = \left(nb\ Tx^{(i)} == 1\right) \tau_6$$

Figure 5. Bus Interconnection Model - Node Automaton

The passage from idle to processing state and vice-versa are local transitions that not affected at all by the other automata, *i.e.*, local events with constant rates are used. However, this is not the case of the passages from idle state to receive and transmitting states. The passage to the transmitting state is granted by means of a local event with a functional rate f_1. This rate has a nonzero value (τ_5), *i.e.*, it may occur, only when no other node is transmitting, *i.e.*, the number of automata in state $Tx^{(i)}$ is equal to zero:

$$f_1 = \left(nb\ Tx^{(i)} == 0\right) \tau_5$$

The passage to the receiving state is modeled analogously. Functional rate f_2 grants the passage to receiving state only if there is one automaton in transmitting state:

$$f_2 = \left(nb\ Tx^{(i)} == 1\right) \tau_6$$

After transmitting or receiving, the return to the idle state will not be affect by any restriction, and, once again, it can be represented by local events with constant rates.

3.1.1. Extension to Multiple Buses

Although most cluster architectures based on bus interconnection have only one bus, architecture with more than one bus could be easily represented by changing the function f_1 to grant access to a bus when at least one bus is available, *i.e.*, considering B the number of buses:

$$f_1 = \left(nb\ Tx^{(i)} < B\right) \tau_5$$

and changing f_2 to grant access when at least one node is transmitting:

$$f_2 = \left(nb\ Tx^{(i)} > 0\right) \tau_6$$

3.2. Ring Cluster

Considering a token ring architecture illustrated by Figure 6, the main difference is the absence of competition for the interconnection media. In this paper it will be assumed an one-way ring with single token, but small model changes could easily represent a different number of tokens and two-way message passings.

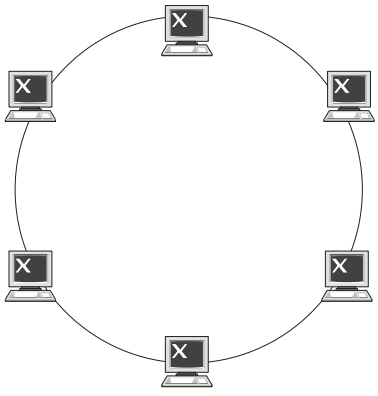


Figure 6. Ring Interconnection Architecture

The modeling technique is similar in the bus model example, since an automaton is used to represent each node. However, the synchronized events are now necessary to describe the synchronization between each pair of neighbor nodes. Each automaton (Figure 7) will have five states, since an additional state ($TxRx^{(i)}$) will be included to represent a node receiving by one interface and, at the same time, transmitting by its other interface.

Each automaton will have four synchronized events to represent the beginning and ending of connections with its neighbors (receiving from previous and transmitting to the next node). Arbitrarily, it will be assumed the start of

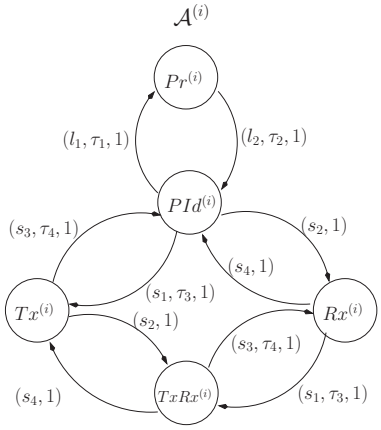


Figure 7. Ring Interconnection Model - Node Automaton

transmition (event s_1) being master in the automaton $\mathcal{A}^{(i)}$ and slave in the automaton $\mathcal{A}^{(i+1)}$, as well as for the event of ending of the transmition (event s_3). For the events concerning reception (event s_2 and s_4), it will be assumed to be slave in automaton $\mathcal{A}^{(i)}$ and master in automaton $\mathcal{A}^{(i-1)}$.

3.2.1. Alternative Modeling for Ring Interconnection

An alternative model to represent ring architectures can use a pair of automata to represent each node (Figure 8). In this case each automaton will represent one of the interfaces of a node. This approach considers the same synchronizing technique as in the original model. In fact, it only splits the interfaces in two automata. The combined states of this pair of automata (nine states) are equivalent to the five states of the previous model, since the processing state in both automata are not compatible with any other state.

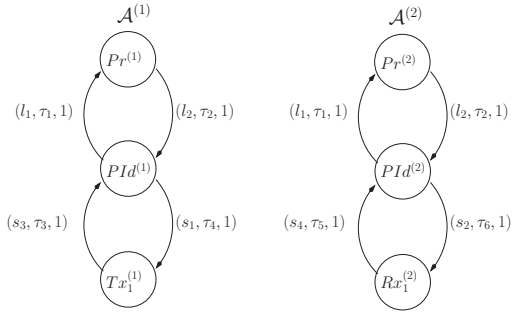


Figure 8. Alternative Ring Model - A Pair of Automata to Each Node

Eventhough, this alternative model is absolutely equivalent to the original model, this concept is important to understand the approach used to model the torus architecture presented in the next section.

3.3. Torus Cluster

The last model proposed represents a *Torus* architecture. Torus topology can be viewed as a two-dimensional ring, where each node is connected to two vertical and two horizontal neighbors (Figure 9). The same consideration about one or two-direction ring is made for this case, *i.e.*, in this paper one-directional rings will be considered, but small changes in the model can be included.

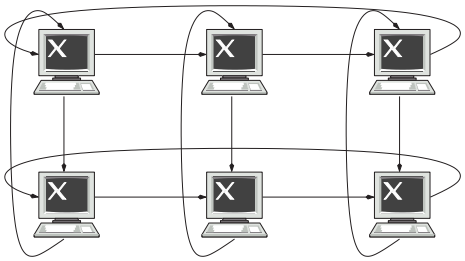


Figure 9. Torus Interconnection Architecture

The modeling technique in this case could be from an automaton to each interface (four automata to each node) to a single automaton to each node. However, the proposed model will consider two automata to represent each node: one to represent its transmitions, and other to represent its receptions (Figure 10). The automata will indicate processing and idle states of node (first two states) and the number of simultaneous transmition (or receptions). It is important to stress out that this representation of a node using two automata intends to facilitate the modeling of the possible number of states of a node.

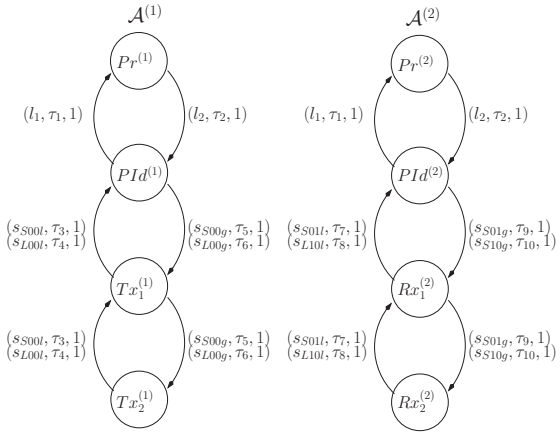


Figure 10. Torus Model - A Pair of Automata to Each Node

The automaton representing the transmitions will have the following four states:

- $Pr^{(1)}$ - the node is processing a task;
- $PId^{(1)}$ - no interface is transmitting a message;
- $Tx_1^{(1)}$ - one interface is transmitting a message;

- $Tx_2^{(1)}$ - two interfaces are transmitting a message;

Analogously, the automaton representing the receptions will have the following states:

- $Pr^{(2)}$ - the node is processing a task;
- $PId^{(2)}$ - no interface is receiving a message;
- $Rx_1^{(2)}$ - one interface is receiving a message;
- $Rx_2^{(2)}$ - two interfaces are receiving a message;

As in the alternative model of the previous section, the combined automaton equivalent to the pair of automata of Figure 10 has some unreachable states. Excepting the processing state in both automata, the three remaing states in each automaton are compatible to each other. It results in ten states (processing state plus the nine combinations -3×3- of the remaining states) reachable of the sixteen possible combinations (4×4).

4. Conclusion

The presented models are very intuitive and their utility is strictly bound to an extensive work mapping the clusters numerical information into the rates of the SAN models. In fact, the models presented in this paper intend to express only the relationship of the interconnection policy to the basic resource sharing restrictions. It was not in the scope of this paper to analyze real cases, nor even to propose a technique to map quantitative information in the models. It was merely intended to show how cluster architectures could be modeled using SAN. Keeping this goal in mind, it is reasonable to argue that the models of the choose architectures will probably satisfy the modeling needs to the performance evaluation of cluster architectures. The logical next steps of this work will be the comparison between the predictions of the analytical models with real clusters configurations.

SAN formalism usually offers more efficient solutions than other traditional methods, like straigh-forward Markov chains, but in the scope of this paper the modeling qualities of SAN are more relevant. The presented models are modular, and therefore can be easily extended to as many nodes as wanted. Unfortunately, to quantitative analysis purposes the limit imposed by the state space explosion can not be ignored. The size of a SAN model is the product of the size of each automaton ($\sum_{i=1}^{N} n_i$). PEPS software running on a average work station (1 Mb of memory) may solve problems 30,000,000 states. For models of bus clusters (Section 3.1), it does not present much of a problem, since PEPS may solve models of 12 nodes. However, models of ring interconnections already may have to stop with models with 10 nodes. These limits are imposed by memory requirements, therefore technological evolutions may facilitate the solution of larger problems. Another possible way, of course, may be the parallelization of PEPS software in order to solve larger SAN models. According to this point of view, such parallelization can also be included in the future work ideas of this paper.

References

[1] A.Benoit, B.Plateau, W.J.Stewart. “Memory efficient iterative methods for stochastic automata networks”. INRIA, Rapport de Recherche No. 4259, 2001. Anonymous ftp *ftp://ftp.inria.fr/INRIA/Publication/RR/RR-4259.ps.gz*

[2] R. Buyya. “High Performance Cluster Computing - Architectures and Systems”. Vol. 1. Prentice Hall, 1999.

[3] M.Davio. “Kronecker products and shuffle algebra”. *IEEE Transactions on Computers*, Vol. C-30, No. 2, pp. 116-125, 1981.

[4] P.Fernandes, B.Plateau, W.J.Stewart. “Efficient descriptor-vector multiplication in stochastic automata networks”. *Journal of the ACM*, Vol. 45, No. 3, pp. 381-414, 1998.

[5] P.Fernandes. “Méthodes numériques pour la solution de systèmes Markoviens à grand espace d’états”. INPG, Grenoble, 1998. (Ph.D. thesis)

[6] K. Hwang, Z. Xu. “Scalable Parallel Computing - Technology, Architecture and Programming”. WCB/McGraw-Hill, 1998.

[7] B.Plateau, K.Atif. “Stochastic automata networks for modelling parallel systems”. *IEEE Transactions on Software Engineering*, Vol. 17, No. 10, pp. 1093-1108, 1991.

[8] B.Plateau. “De l’Evaluation du Parallélisme et de la Synchronisation”. Paris-Sud, Orsay, 1984. (Ph.D. thesis)

[9] W.J.Stewart “Introduction to the Numerical Solution of Markov Chains”. Princeton University Press, 1994.

[10] PEPS Project: Performance Evaluation of Parallel Systems. *http://www-apache.imag.fr/software/peps/*