Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 232 (2009) 165–178
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Modelling Grid5000 point availability with SAN ⋆

Leonardo Brenner [a,1], Paulo Fernandes[b], Jean-Michel Fourneau[c] and Brigitte Plateau[a]

[a] *Laboratoire LIG, CNRS-INRIA-INPG-UJF, 51, av. Jean Kuntzmann 38330 Montbonnot Saint-Martin France*

[b] *PUCRS, Faculdade de Informática, Av. Ipiranga, 6681 90619-900 Porto Alegre, Brazil*

[c] *PRiSM, Université de Versailles Saint-Quentin-en-Yvelines, 45, Av. des Etats-Unis, 78035 Versailles, France*

Abstract

The point availability is the probability that a system is available at time t. As grids need a lot of resources to be really operational, the evolution of their availability with time or depending on a maintenance process is a hot topic. But as grids contain a lot of resources, it is quite difficult to model and solve this problem. Here we advocate a component based description which is associated to a tensor based numerical approach.

Keywords: Stochastic Automata Networks, grid computing, point availability.

1 Introduction

Grid computing can offer a high performance computing environment but can also suffer from severe overheads introduced by unreliable infrastructure, low performance middleware and cross domain connectivity. Grid computing systems are different from conventional distributed computing systems by their focus on large-scale resource sharing, where processors and communication have significant influence on reliability [7]. Thus, the performance and reliability analysis of grids may have a high impact on these new computing environments which are becoming more and more frequent nowadays.

Reliability and dependability analysis of grids may use state-based (typically Markov chains) and non-state-based models. Non-state-based techniques such as

⋆ This work was supported by project *Sure-Paths* from ACI Sécurité and ANR "programme blanc" project *SMS*

[1] Author receives grants from CAPES-Brazil - (BEX 2222-03)

doi:10.1016/j.entcs.2009.02.056

Reliability Block Diagrams or Fault Trees [2] mainly rely on independence assumptions between components. Such assumptions are not usually true for grids where not only faults and errors, but also repairs are dependent. Markov chains have been used for a long time to model availability, performability and dependability of complex networks and computer systems. However, this approach suffers initially from two important drawbacks. First, it is difficult to obtain the states, the transitions and the probabilities from the description and the specifications of the system. But the second problem is even tricky. It is really difficult to obtain the analytical solution for the steady-state or the transient distribution of the chain. The memory space requirements and the time complexity are often too important to solve the chain for real problems [17].

Here we are interested in the point availability $A(t)$ which is defined as the summation of elementary reward functions $r(i)$ on the transient distribution $\pi(t)$, *i.e.*, $A(t) = \sum_i r(i)\pi_i(t)$, where $r(i)$ is 1 when the system is UP and 0 when it is DOWN. Thus the numerical computation is mainly the computation of the transient distribution and the summation of the elementary rewards $r(i)$ to obtain $A(t)$.

Steady-state availability, *i.e.*, the limit of $A(t)$ when t goes to infinity, can be computed for very large systems because it relies on the numerical analysis of steady-state distribution and very efficient techniques are known [6]. Point availability is much harder. The algorithms are simpler to write but the number of iterations may be very important and the size of the Markov chain has a large impact on the feasibility of the analysis.

Modelling a grid platform such as Grid'5000 is a difficult task. A grid needs a large number of disks, CPUs and network links to be fully operational. The Markov Chain to describe all these resources usually has an extremely large state space that we cannot be handled even in a sparse format. Unfortunately, it is not easy to simplify such huge state space since the rewards functions take into account the whole description of the resources. However, a point availability analysis can be achieved as we show in this paper. Typically we will define an operational state as a state where more than $X\%$ (X will be called later threshold) of the CPUs are available and interconnected by the network. The typical results we can extract from such a transient analysis is of the following type: given an initial state (which is operational but including some failed components) we can observe the evolution of the grid under different repair policies, and thus compare these policies. A deeper investigation would allow to choose, for a given initial state, the best repair policy for the grid.

Stochastic Automata Networks is one of the many high level formalisms which have been designed to overcome the state space explosion problem using a tensor representation of the chain. Instead of storing the chain as a sparse matrix, it is possible to derive from the specifications some small matrices which are used to build the transition matrix of the chain [14]. Also, computation can be handled on the reachability states space, which may be much smaller than the potential states space [1].

$$Q = \sum_i \bigotimes_j Q_i^j$$

Many high level formalism such as Petri nets and Stochastic Process Algebra are now able to store the transition matrix Q of a chain in a tensor form [11,13]. However, new methods have to be derived to take advantage of this tensor representation [12,16] and they may be more complex than simple algorithms based on sparse matrix representation. Typically the algorithms [9,10,15] to compute the point availability are based on the uniformization of the continuous time Markov chain to control the truncation error and on a (vector matrix vector) multiplication. Here we advocate tensor representation of the chain as an interesting alternative to obtain, store and analyze the chain and the point availability of systems. Using new version of classic algorithms tailored for tensor representation (implemented in the last version of PEPS [4]), we are able to analyze the grid point availability problem we want to cope with.

The paper is organized as follows: Section 2 is devoted to a brief introduction to SAN modelling. A grid model and a study case are presented in Section 3 and 4. The numerical results are reported in Section 5.

2 SAN Formalism

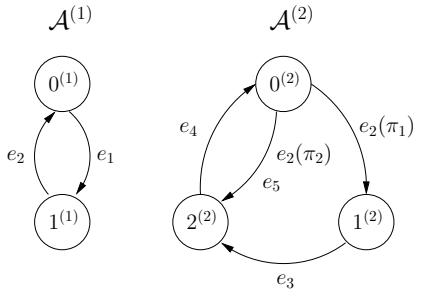
Stochastic Automata Networks (**SAN**) is a structured Markovian formalism, *i.e.*, it describes continuous-time Markovian models not as a flat system, but as a structured (modular and organized) collection of subsystems. The basic modeling principle of **SAN** is to describe a whole system by a collection of subsystems with an independent behavior and occasional interdependencies. 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 always build a continuous-time stochastic process related to **SAN** [17,5].

The *global state* of a **SAN** model is defined by the cartesian product of the *local states* of all automata. There are two types of events that change the global state: *local* and *synchronizing events*. Local events change the **SAN** global state passing from a global state to another that differs only by one local state. Synchronizing events can change simultaneously more than one local state, *i.e.*, two or more automata can change their local states simultaneously. In other words, the occurrence of a synchronizing event forces all concerned automata to fire a transition corresponding to this event.

Each event is represented by an *identifier* and a *rate* of occurrence, which describes how often a given event will occur. Each transition may be fired as result of the occurrence of any number of events. In general, non-determinism among possible different events is dealt with according to Markovian behavior, *i.e.*, any of the events may occur and their occurrence rates define the relative frequency with which each of them will occur. However, if, from a given local state, the occurrence of a given event can lead to more than one state, then an additional *routing prob-*

ability must be informed to each possible destination state. The absence of routing probability is tolerated if only one transition can be fired by an event from a given local state.

The other possibility of interaction among automata is the use of functional rates. Any event occurrence rate may be expressed by a constant value or a function of the state of other automata. By contrast with synchronizing events, functional rates are one-way interaction among automata, since it affects only the automaton where it appears.



Type	Event	Rate
loc	e_1	f
syn	e_2	μ
loc	e_3	σ
loc	e_4	δ
loc	e_5	τ

$$f = \left[\left(st\ \mathcal{A}^{(2)} == 0^{(2)}\right) * \lambda\right] + \left[\left(st\ \mathcal{A}^{(2)} == 2^{(2)}\right) * \gamma\right]$$

Figure 1. Example of a **SAN** model

Figure 1 presents a **SAN** model with two automata, one synchronizing event (e_2) with a constant rate, and four local events, being three with constant rates (e_3, e_4 and e_5) and one with a functional rate (e_1). In this model the rate of the event e_1 is a functional rate f semantically explained below, and described inside Figure 1 using the **SAN** notation. The interpretation of such a function can be viewed as the evaluation of an expression of non-typed programming languages, *e.g.*, C language, where each comparison is evaluated to value 1 (*true*) or value 0 (*false*).

$$f = \begin{cases} \lambda \text{ if automaton } \mathcal{A}^{(2)} \text{ is in the state } 0^{(2)} \\ 0 \text{ if automaton } \mathcal{A}^{(2)} \text{ is in the state } 1^{(2)} \\ \gamma \text{ if automaton } \mathcal{A}^{(2)} \text{ is in the state } 2^{(2)} \end{cases}$$

The firing of the transition from states $0^{(1)}$ to $1^{(1)}$ occurs with rate λ if automaton $\mathcal{A}^{(2)}$ is in state $0^{(2)}$, or γ if automaton $\mathcal{A}^{(2)}$ is in state $2^{(2)}$. If automaton $\mathcal{A}^{(2)}$ is in state $1^{(2)}$, the transition from states $0^{(1)}$ to $1^{(1)}$ does not occur (rate equal to 0). It is important to observe that the use of functions allows a compact and flexible way to describe in one single event (local or synchronized) alternative behaviors [5].

Figure 2 shows the equivalent Markov chain model of the **SAN** model in Figure 1. Assuming the state $0^{(1)}0^{(2)}$ is an initial state, only 5 of the 6 states in this Markov chain model are reachable. In order to express the reachable global states of a **SAN** model, it is possible to define a (*reachability*) function. The reachable states could also be computed by analyzing all possible firing sequences, starting from a given reachable initial state. For the model in Figure 1, the reachability function excludes the global state $1^{(1)}1^{(2)}$, thus:

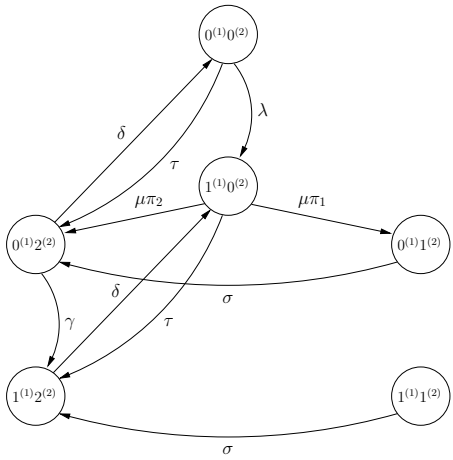


Figure 2. Equivalent Markov chain model of the **SAN** model in Figure 1

$$Reachability = ! \left[\left(st\ \mathcal{A}^{(1)} == 1^{(1)} \right) \&\& \left(st\ \mathcal{A}^{(2)} == 1^{(2)} \right) \right]$$

One of greatest advantages of the **SAN** formalism in comparison with straightforward Markov chain, and even other structured formalism is to have since its first definition [14] a compact form to store the infinitesimal generator of the equivalent Markov chain. Instead of storing an (usually) huge matrix, the **SAN** formalism defines a storage based on a tensor formula of considerably smaller matrices. Tensor, or Kronecker, algebra [8,12] is defined as a set of multi-dimensional structures (tensors) and algebraic operations. It usually allows the very compact description of quite large and complex matrices. Also, computation can be handled without ever generating extensively the equivalent Markov chain.

3 SAN Model for Grid Platforms

This section presents a possible description of a grid platform using **SAN**. Basically, a grid platform is composed of a set of interconnected sites. Each site provides some CPUs. We propose in this paper a simple way to model a grid platform composed of S sites and L connections using only two kinds of generic automata. The first kind of automaton represents the sites of the grid. This kind of automaton is called *site* automaton. The second kind of automaton, called *link* automaton, represents the direct connections between two sites, *i.e.*, each *link* automaton models a direct connection.

The $Site^{(i)}$ automaton (Figure 3) describes the number of CPUs ($N^{(i)}$) provided by the i^{th} site.

Each state represents the number of *UP* CPUs in the site ($N^{(i)}$ represents all CPUs *UP* and 0 none CPU *UP*). The event fp_i models a CPU failure and the event rp_i models CPU repairs in the site i. CPU failure event fp_i is independent and it has constant rate (λ_i). CPU repair event rp_i has functional rate $\mu_i.f_i$ that depends on the global state of the grid. The rate μ_i represents the effective rate to repair a CPU (the inverse of the time to repair it), while the function f_i models the

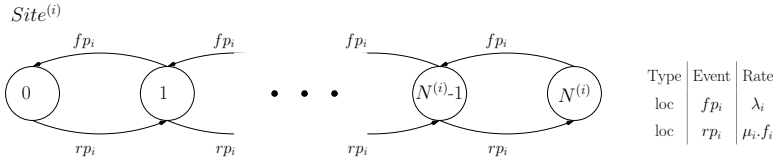


Figure 3. Generic *site* automaton

grid state dependence. The function f_i associated to the repairing rate allows us to model different repair behaviors, like priority repair ordering. This feature will be shown in the case study of Section 4 where different functions are used to change the repair service policies. Here we use specifically the ability of **SAN** formalism to model dependencies among components using functions. Typically, function f_i which determines rates of automaton $Site^{(i)}$ has as argument the current state of automata $Link^{(l)}$ which are connected to this site.

The $Link^{(l)}$ automaton (Figure 4) describes the state of a site-to-site connection. A *link* automaton has only two states: *up* and *down*.

The event $dlink_l$ characterizes a failure of the l^{th} link. This event occurs independently of the state of the site and has a constant rate γ_l. In contrast, the event $ulink_l$ characterizes a repair of the l^{th} link and, it depends on the network state and has a functional rate $\beta_l.g_l$. Once again, β_l defines the inverse of the time spent to repair the connection l and the function g_l models the grid state dependence and repair priority. The events $dlink_l$ and $ulink_l$ characterize a simple failure and repair of a connection. However, a second kind of failure/repair can occur. This kind of failure/repair is a local network failure/repair. In this case, all connections of a site become *down*. Analogously, when the local network is repaired, all connections return to *up*. Local network failures are modeled by synchronized events $dsite_i$ and the corresponding repairs by events $usite_i$. The events $dsite_i$ have constant rate τ_i. The repairing events $usite_i$ are also dependent of the grid state and have functions rates $\rho_i.p_i$.

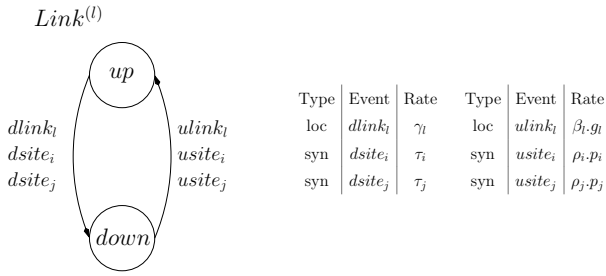


Figure 4. Generic *link* automaton

If a simple connection failure occurs (event $dlink_l$) the site can still be accessible by others connections, if it exists at least one connection in state *up*. Otherwise, *i.e.*, if a local network failure occurs ($dsite_i$) the site CPUs are inaccessible to all other sites.

4 Case Study: Grid'5000

Grid'5000 project is a highly reconfigurable, controllable and monitorable experimental platform dedicated to grid research. This project proposes the creation of a platform with 5000 CPUs as an experimental testbed for research community [3].

Grid'5000 is geographically distributed over 9 sites in France (Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia-Antipolis, and Toulouse). Each site (local platform) provides at least one cluster with 100 or more nodes, where 2/3 are dual CPUs. Each site is connected with at least two others sites by the RENATER Education and Research Networks with 10GB/s links. Figure 5 shows the sites connections network.

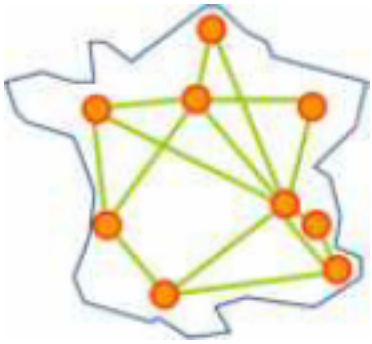


Figure 5. Grid'5000 Site Connections

4.1 Proposed Model

To validate the ideas proposed in Section 3, we modelled the Grid'5000 platform using **SAN** formalism. Unfortunately, the whole model including all Grid'5000 sites (9 automata) and all direct connections (15 automata) results in a Markov chain with a product space state of 1.3×10^{22} states, which is not practically solvable (in this model, all states are reachable).

However, some Grid'5000 characteristics can be explored to reduce the problem size without compromising the performance indices accuracy. An important characteristic to be explored is the redundant connections, *e.g.*, two or more direct physical connections to link two sites. This redundancy results in an insignificant failure probability. In fact, this Grid'5000 characteristic allows us to consider a set of sites like one grouped *secure* site and ignore the internal *secure* site connections.

Applying this grouping procedure for Grid'5000 platform, we can transform 9 physical sites in 4 *secure* sites and reduce the number of connections to 9 connections between *secure* sites. The 4 *secure* sites are: OLR composed by Orsay, Lille, and Rennes with 1406 CPUs; LGS composed by Lyon, Grenoble, and Sophia-Antipolis with 886 CPUs; TB for Toulouse and Bordeaux with 700 CPUs; and $Nancy$ composed by just one site, Nancy, with 334 CPUs. These numbers of CPUs in each site are valid at the date January, 25^{th}, 2008.

After grouping a set of sites in only one *secure* site, some connections become redundant too, *i.e.*, the connections between Orsay-Lyon, Rennes-Lyon, and Lille-Lyon. These connections can be grouped in only one connection with a reduced failure rate. A grouped connection failure rate is easily computed by the combination of individual connections failure rates. This procedure reduces the number of direct connections to 5.

Another reduction can be applied based on clusters characteristics. We can assume that processors are assembled in groups, and the CPUs failures occur by blocks. This assumption is very reasonable, especially if we consider that many CPUs are inside a same physical machine. We consider in our model blocks of 32 CPUs.

Applying all reductions, the final model has 4 *site* automata (OLR, LGS, TB, and $Nancy$) and 5 *link* automata ($OLR-LGS$, $OLR-TB$, $OLR-Nancy$, $LGS-TB$, and $LGS-Nancy$) and the model drops to the reasonable size of 9×10^6 states.

To test different scenarios, we assume some constraints. We have one local repairman per site and a global repairman. The local repairman is in charge of repairing the local CPUs and the local network failures. The global repairman is in charge of repairing site-to-site connections.

Keeping in mind these constraints, we propose three different basic configurations for the Grid'5000 platform. The first two configurations (Sections 4.1.1 and 4.1.2) use exactly the same grid platform but with different repair service policies. These different repair policies are modeled by functions associated to the repair events. The third configuration (Section 4.1.3) is a variation of the first one considering less repairing options.

4.1.1 Local Network Priority

In the first configuration, the local connection repairs have priority over CPU local repairs. No CPU repairs are executed if there exists a local network failure. This priority rule is modelled using the functions associated with the site's CPU repair service. In our model, we have the functions f_{OLR}, f_{LGS}, f_{TB}, and f_{Nancy} associated, respectively, to the repair events in OLR, LGS, TB, and $Nancy$ automata sites. These functions are described below:

$$f_{OLR} = ((st\ OLR{-}LGS == up)\ ||\ (st\ OLR{-}TB == up)\ ||\ (st\ OLR{-}Nancy == up))$$

$$f_{LGS} = ((st\ OLR{-}LGS == up)\ ||\ (st\ LGS{-}TB == up)\ ||\ (st\ LGS{-}Nancy == up))$$

$$f_{TB} = ((st\ OLR - TB == up)\ ||\ (st\ LGS - TB == up))$$

$$f_{Nancy} = ((st\ OLR - Nancy == up)\ ||\ (st\ LGS - Nancy == up))$$

These functions return 0 or 1. Thus they enable (or not) the repair event. According to the repair policy, a function of the site i ($i = OLR, LGS, TB, Nancy$) is equal to 0 only when all connections of this site are down.

In this configuration, all functions p_i associated to the $usite_i$ events are constant and equal to 1 because the local network repairs have priority over CPUs repairs. Then, no restriction is imposed to the $usite_i$ events occurrence.

We assume no repair priority rules among the site-to-site connections. However, the repair rate depends on the number of *down* connections: the repairman time is shared among all *down* connections. To model this behavior, we multiply the repair rates by the function:

$$g_l = \frac{1}{nb\ \ [OLR - LGS..LGS - Nancy]\ \ down}$$

This function computes the inverse of the number of connections in state *down*.

4.1.2 Connections Priority

The second configuration keeps the local network repair priority over CPU repair and additionally, a priority order for site-to-site connection repairs is included, *i.e.*, if two connections are *down*, the repairs are executed in a predefined order. In this configuration, the decreasing priority order is $OLR - LGS$, $OLR - TB$, $OLR - Nancy$, $LGS - TB$, and $LGS - Nancy$. To model this priority policy, we associate to each connection repair event a priority function (g_l). The priority functions must analyze the current state of *all* components having higher priority connections. These priority functions return 1 if the state of all highest priority connections are *up*, 0 otherwise. For instance, the function associated to the second priority connection ($OLR - TB$) must analyze the first priority connection state ($OLR - LGS$) [2]. This function is expressed by:

$$g_{OLR-TB} = (st\ OLR - LGS == up)$$

The others priority functions are:

$$g_{OLR-LGS} = 1$$

$$g_{OLR-Nancy} = (st\ OLR - LGS == up)\ \&\&\ (st\ OLR - TB == up)$$

$$g_{LGS-TB} = (st\ OLR - LGS == up)\ \&\&\ (st\ OLR - TB == up)\ \&\&\ (st\ OLR - Nancy == up)$$

[2] The highest priority has the priority function equal to 1.

$$g_{LGS-Nancy} = (st\ OLR - LGS == up)\ \&\&\ (st\ OLR - TB == up)\ \&\&$$
$$(st\ OLR - Nancy == up)\ \&\&\ (st\ LGS - TB == up)$$

4.1.3 Degraded Local Network Priority

The third configuration is based on the first configuration (Local Network Priority). Nevertheless, in this new configuration, we remove one site-to-site connection. This *degradation* in the model aims at modelling a long repair service, *i.e.*, a physical cable replacement. To model this situation, the connection between OLR and LGS was removed. This configuration allows us to test the point availability of the Grid'5000 platform during a long repair service.

5 Grid'5000 Point Availability

In the previous section, we presented three basic configurations modelling Grid'5000 availability. These configurations were solved using PEPS tool. Table 1 shows the failure and repair rates. They are expressed as rates of failures/repairs per day.

CPU failure/repair rates

Site	failure	repair
OLR	0.14	2.0
LGS	0.15	2.0
TB	0.13	2.0
Nancy	0.16	2.0

Connections failure/repair rates

Connection	failure	repair
OLR-LGS	0.015	1.0
OLR-TB	0.014	1.0
OLR-Nancy	0.013	1.0
LGS-TB	0.016	1.0
LGS-Nancy	0.014	1.0

Local network failure/repair rates

Site	failure	repair
OLR	0.030	2.0
LGS	0.029	2.0
TB	0.030	2.0
Nancy	0.031	2.0

Table 1
Rates for the Grid'5000 **SAN** model

To test different situations, we computed the point availability of the three basic configurations presented above with three different thresholds: for the less demanding threshold, we need at least 55% of CPUs *UP* to consider the system available. The second threshold was fixed at 75% and the third one at 95%.

In our tests, we start with all CPUs *UP* but 2 connections *down*. The *down* connections are $OLR - TB$ and $OLR - Nancy$. Notice that for the *Degraded Local Network Priority* configuration, the connection $OLR - LGS$ does not exist and, in this case, the CPUs of OLR are not accessible at the beginning. This particular initial configuration, where the system is not fully operational, was chosen to better analyze the performance of different repair services. This analysis is not possible when we start the tests with a fully operational system as no repair in needed. An exhaustive analysis would require to perform this analysis for a set of "well-chosen" such initial states.

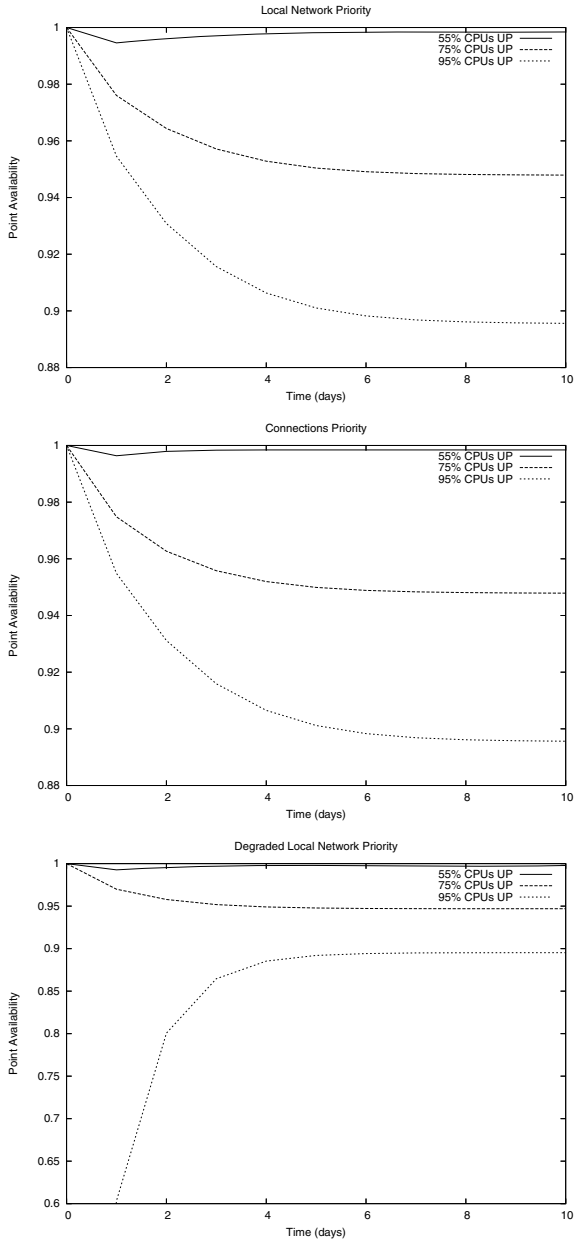


Figure 6. Grid'5000 Point Availability for different thresholds

In Figure 6, the graphs show different thresholds for each configuration. We can observe three different behaviors. The first one is observed with a low threshold (55%). This behavior shows that, when the availability measure is not demanding (here we measure the probability that more than 55% of the CPUs are available) the 3 policies are equally efficient.

Now let's consider the behavior of the threshold 75% for all configurations: at initial time, 2 connections are *down* and all CPUs are running. As time passes, these

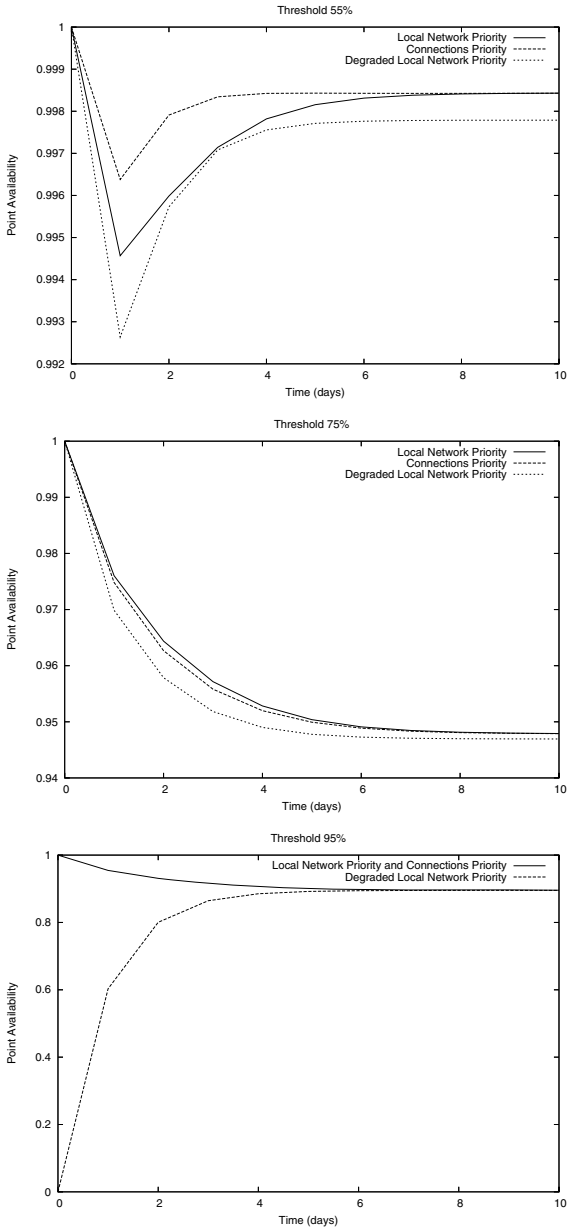


Figure 7. Grid'5000 Point Availability for different configurations

curves show that the number of CPUs not available (either *down* or not reachable) increases showing a degradation of the grid. For the threshold 95%, we observe the same behavior as for 2 configurations, but a different one for the "Degraded Local Network Priority" configuration. In this case, a *down* connection breaks the grid in two disconnected parts, where no part is able to provide satisfactorily the system demand even with a high number of *UP* CPUs. The demanding threshold of 95% implies that the repair of the missing connections will provoke an increase of the number of available CPUs. This phenomenon is visible for the threshold 95% and

was not visible for the threshold 75%.

The graphs in Figure 7 allow to determine the best repair policy according the system threshold requirement. "Degraded Local Network Priority" policy is always the worst policy for all thresholds and it shows the importance of redundant connections. For the threshold 55% the best repair policy is "Connections Priority", followed by "Local Network Priority". However, for the threshold 75%, the best policy is "Local Network Priority", followed by "Connections Priority". For the 95% threshold, "Local Network Priority" and "Connections Priority" have a very close performance, and their lines are indistinguishable in the graph. This behavior shows that, for high demanding threshold, these two repair policies are equivalent. For this threshold, "Degraded Local Network Priority", the bad results are explained by the system configuration where at time 0, we have a disconnected grid.

The measures extracted from the model can be used to determine the best repair policy and consequently a better resource allocation facing a multiple failure problem.

6 Conclusion

In this paper, we proposed a simple technique to model a grid platform and to extract some performance indices. Starting from a large grid platform, we could build a tractable **SAN** model using only two kinds of generic automata and functional rates, and thus provide performance indices on the Grid'5000 platform point availability.

Section 5 shows point availability information for three basic configuration models using different thresholds. The goal of our tests was not to show a good or bad Grid'5000 platform management but to test and compare different scenarios. From these we can expect that a well-parametrized model would provide a deep and thorough analysis of the grid management options.

We advocate in this paper the use of a high level stochastic structured formalism (**SAN**) as a quick and low cost mean to model and test various grid connections with different repair service policies. The use of **SAN** formalism and Generalized Tensor Algebra (GTA) allows us to describe and solve quite efficiently large grid platforms. To illustrate this point, the time spent to solve the models was 93.3 minutes (528 iterations) for the smallest and more rapidly solved model ("Degraded Local Network Priority") and up to 207.9 minutes (573 iterations) for the largest model. Overall, the analysis of three grid configurations considering three different thresholds was performed in little more than a day (approx. 25.5 hours) running the models with a Xeon 1.6GHz with 1GB of memory running Fedora Core 7.

As said before, this paper offers some transient analysis based on point availability values. However, other performance indices can be extracted from the same model in order to provide a powerful tool to predict availability of a rather complex grid without interfering with the actual grid operation. A careful analysis of performance indices of the proposed model may offer unexpected solutions to possible problems in the grid management and help to plan maintenance services with a

more than just hints.

References

[1] Benoit, A., L. Brenner, P. Fernandes, B. Plateau and W. J. Stewart, *The PEPS Software Tool*, in: *Computer Performance Evaluation / TOOLS 2003*, LNCS **2794** (2003), pp. 98–115.

[2] Bolch, G., S. Greiner, H. de Meer and K. S. Trivedi, "Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications," John Wiley & Sons, 1998.

[3] Bolze, R., F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E. Talbi and I. Touche, *Grid'5000: a large scale and highly reconfigurable experimental grid testbed.*, International Journal of High Performance Computing Applications **20** (2006), pp. 481–494.

[4] Brenner, L., P. Fernandes, B. Plateau and I. Sbeity, *Peps2007 - stochastic automata networks software tool*, in: *QEST '07: Proceedings of the Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007)* (2007), pp. 163–164.

[5] Brenner, L., P. Fernandes and A. Sales, *The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations*, in: 20^{th} *Annual UK Performance Engineering Workshop*, Bradford, UK, 2004, pp. 48–60.

[6] Czekster, R. M., P. Fernandes, J.-M. Vincent and T. Webber, *Split: a flexible and efficient algorithm to vector-descriptor product*, in: *VALUETOOLS*, 2007, pp. 83–95.

[7] Dai, Y. S., M. Xie and K. L. Poh, *Reliability analysis of grid computing systems*, in: *PRDC '02: Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing* (2002), pp. 97–104.

[8] Davio, M., *Kronecker Products and Shuffle Algebra*, IEEE Transactions on Computers **C-30** (1981), pp. 116–125.

[9] de Souza e Silva, E. and H. R. Gail, *Calculating cumulative operational time distribution of repairable computer systems*, IEEE Transactions on Computers **35** (1986), pp. 322–332.

[10] de Souza e Silva, E. and H. R. Gail, *Calculating availability and performability measures of repairable computer systems using randomization*, Journal of the ACM **36** (1989), pp. 171–193.

[11] Donatelli, S., *Superposed generalized stochastic Petri nets: definition and efficient solution*, in: R. Valette, editor, *Proceedings of the* 15^{th} *International Conference on Applications and Theory of Petri Nets* (1994), pp. 258–277.

[12] Fernandes, P., B. Plateau and W. J. Stewart, *Efficient descriptor - Vector multiplication in Stochastic Automata Networks*, Journal of the ACM **45** (1998), pp. 381–414.

[13] Hillston, J. and L. Kloul, *An Efficient Kronecker Representation for PEPA models*, in: L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop)* (2001), pp. 120–135.

[14] Plateau, B., *On the stochastic structure of parallelism and synchronization models for distributed algorithms*, in: *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems* (1985), pp. 147–154.

[15] Rubino, G. and B. Sericola, *Interval availability distribution computation.*, in: *23th IEEE International Symposium on Fault Tolerant Computing*, Toulouse, France, 1993, pp. 48–55.

[16] Stewart, W., K. Atif and B. Plateau, *The numerical solution of stochastic automata networks*, European Journal of Operational Research **86** (1995), pp. 503–525.

[17] Stewart, W. J., "Introduction to the numerical solution of Markov chains," Princeton University Press, 1994.