GridSPN: a Grid-based non Markovian Petri Nets Tool

S. Distefano, A. Puliafito and M. Scarpa
Università di Messina, Dipartimento di Matematica
Contrada Papardo, S. Sperone, 98166 Messina, Italy
salvatdi[apulia, mscarpa]@ingegneria.unime.it

Abstract

This paper describes the new features implemented in WebSPN, a modeling tool for the analysis of non-Markovian stochastic Petri nets. WebSPN adopts a discretization of time and an approximation of non-exponentially distributed firing time transitions by means of the Phase type distribution. To improve the MPICH parallel implementation of this algorithm, in this paper we describe the porting of WebSPN from the MPI to the Grid computational paradigm. Besides a better flexibility in accessing computational and storage resources, one of the main advantages is the introduction of a Fault Recovery System to detect and recover from eventual machine faults. The resulting new tool is named GridSPN.

Keywords: Non-Markovian SPN, DPH, parallel computation, MPI, Grid, MPICH-G2.

1 Introduction

During the past few years, we have worked on a modeling tool for the analysis of non-Markovian stochastic Petri nets (NMSPNs) called WebSPN [3]. This tool provides a discrete time approximation of the stochastic behavior of the marking process [10], which results in the possibility to analyze a wider class of Petri net models. It is based on the generation of DTMCs (Discrete Time Markovian Chains) through a discretization of time. But it suffers of the drawback of state space explosion. One solution to this problem is to formulate the algorithm in parallel terms as we have reported in [7], where MPI [1] is used as the programming environment.

In this paper we propose to adopt a Grid computational model [6, 5] to access distributed computing and storage resources. The Grid will allow to dramatically increase the number of processors used for the analysis, as well as the total amount of storage capacity. By this become possible to adapt the job distribution to the load conditions of the involved machines and to implement a fault tolerance mechanism to detect and recovery eventual faults. We ported the WebSPN tool from MPI to the Grid environment, exploiting the MPICH-G2 [8] to interface the MPICH and the Globus Toolkit 2 (GT2) [2], enriching it with some interesting features like the adaptive load distribution, the security management and, above all, the fault recovery system.

In section 2 the algorithm is briefly described, introducing a complexity estimation used in the adaptive load distribution. Section 3 presents the algorithm implementation in the Grid environment and finally, in section 4, some conclusive consideration are given.

2 Algorithm Overview

WebSPN is a tool for the analysis and solution of non-Markovian SPNs (NMSPNs) models. A NMSPN is a stochastically timed PN where to each timed transition \( t \) is assigned a general random firing time \( \gamma \) with a cumulative distribution function \( G(t) \) and a firing policy among \( \text{prd} \), \( \text{prs} \) and \( \text{pri} \) [4]. A timed transition fires as soon as the age variable \( a \) associated to the timed transition \( t \) reaches the value of the firing time \( \gamma \) [4].

The solution algorithm is based on a discretization of the firing time distributions of the timed transitions, approximated with Discrete PHase type (DPH) [9] distributions, by dividing the time into \( \delta \)-wide time slots. A DPH distribution is described by a matrix; in the following, we will identify with \( B_k \) the matrix describing the DPH associated with transition \( t_k \).

With this representation, the stochastic process modeling the temporal evolution is itself a homogeneous Discrete Time Markov process \( D \), whose states are identified by the net marking and the stage of firing of each transition. The Markov process can be constructed expanding the reachability graph \( R(M_0) \) [10].

The overall algorithm [3] can be schematized in three steps:

1. Reachability Graph Generation (RG): from the NMSPN, specified the initial marking \( M_0 \), the RG is gen-
1. Expansion: each job processes the states included in its work lists, verifying they have not already been generated. The processing is iterated until the work lists are empty or one of the other lists reaches the buffer size limit.

2. Sending: a job sends the states in the transmission lists to the corresponding jobs. Data transmission is done by means of non-blocking calls, thus the job immediately continues with the phase of receiving.

3. Receiving: a job checks whether other jobs have sent messages to it.

4. Termination: the job enters this phase when its input queue remains empty for a time period longer than a fixed timeout. If all of the other jobs are in the same condition, the expansion activity ends.

![Figure 1. Graphical representation of the parallel algorithm.](image)

2.2 Algorithm complexity

To evaluate the total number of arcs produced (the objects that require memory space) a single generic marking $M_i \in \mathcal{R}(M_0)$ is considered. The evaluation is performed starting from the non exponential transitions (MEM) enabled in $M_i$. In the following, $\nu_k$ and $\rho(B_k)$ are the number of states and the non null elements of the matrix $B_k$ (i.e. the arcs of the DPH), used to approximate the MEM transition $t_k$, respectively.

To describe the memory requirements, we need to introduce the following notation: we indicate with $A^P(i)$, $A^S(i)$ and $A^I(i)$ the set of enabled transitions in $M_i$ with a prd, prs and pri memory policy respectively associated. When in marking $M_i$ several prd and prs transitions are enabled, the algorithm produces at most $\prod_{t_k \in A^P(i) \cup A^S(i)} \nu_k$ states [10], since each state stores different values of the age
memory for each enabled transition. The number of arcs is
\[
\prod_{t_k \in \mathcal{A}_{M}^t(i)} \rho(B_k),
\]
corresponding to different possible evolutions of the MEM transitions from states with different values of the age memory.

When a pri transition \(t_k\) is enabled in a marking \(M_i\) \((t_k \in \mathcal{A}_i^1(i))\), \(q^{[k]}\) samples are computed; the \(i\)-th sample produces \(i\) states, as explained in [10], thus the total number of states deriving from this transition is
\[
\sum_{i=1}^{q^{[k]}\,(q^{[k]}+1)/2} i = \frac{q^{[k]}\,(q^{[k]}+1)}{2}.\]
Each state has only one outgoing arc that describes the memory increment of \(t_k\), thus the number of arcs is \(q^{[k]}\,(q^{[k]}+1)/2\) too.

When in marking \(M_i\) a prs transition \(t_k\) is active, but not enabled, a transition \(t_k\) has memory, but no phase change is possible inside the DPH. Therefore, the number of arcs only depends on the enabled transitions in \(M_i\). More precisely, the number of arcs is the number of enabled transitions in \(M_i\) multiplied by the number of states \(\nu_k\) of the prs transition. The same is true for pri transitions, but the number of states is \(q^{[k]}\).

Each arc describes an event that may occur in the interval of size \(\delta\). Up to now, only the events related to MEM transitions are considered, but all the exponential transitions have to be considered too. When \(\gamma(i)\) transitions are enabled in marking \(M_i\), the number of possible firings is \((\gamma(i)+1)\). This quantity is computed considering that either one exponential transition at the time can fire \((\gamma(i)\) events), or none of them can fire.

From these considerations the maximum number of arcs produced by the algorithm in marking \(M_i \in \Omega/\Omega^E\), where \(\Omega\) is the set of all tangible markings and \(\Omega^E\) is the set of states where no MEM transition is active, is:
\[
N(i) = \left( \prod_{t_k \in \mathcal{A}_{M}^t(i) \cup \mathcal{A}_{E}^e(i)} \rho(B_k) \right) \left( \prod_{t_k \in \mathcal{E}^S(i)/\mathcal{A}_{E}^e(i)} \nu_k \right) \left( \prod_{t_k \in \mathcal{E}^L(i)/\mathcal{A}_{E}^e(i)} \frac{q^{[k]}\,(q^{[k]}+1)}{2} \right) \left( \prod_{t_k \in \mathcal{E}^L(i)/\mathcal{A}_{E}^e(i)} q^{[k]} \right) (\gamma(i)+1)
\]
where the sets \(\mathcal{E}^S(i)\) and \(\mathcal{E}^L(i)\) denote the prs and the pri transitions with memory in marking \(M_i\), respectively. The total number of arcs \(L\) thus becomes:
\[
L = \sum_{M_i \in \Omega/\Omega^E} N(i) + \sum_{M_i \in \Omega^E} (\gamma(i)+1) \quad (2)
\]
The quantity in equation (2) is an upper bound on the number of arcs produced by the discretization. The exact number depends on the Petri net and on the DPHs used to approximate the transitions in the net. The amount of memory required by the algorithm is proportional to the number of arcs of the DTMC. Hence, an upper bound for the memory requirement is:
\[
O \left( \sum_{M_i \in \Omega/\Omega^E} N(i) + \sum_{M_i \in \Omega^E} (\gamma(i)+1) \right). \quad (3)
\]

3 Implementation

The parallel implementation of WebSPN has been developed on the MPI infrastructure, through the services and the primitives provided by MPICH. To maintain and respect the previous choices, we have adapted this implementation to the Grid architecture, using MPICH-G2, the latest available Grid-enabled implementation of the interface between the MPICH and the Globus ToolKit 2 (GT2) protocols. This choice allowed us to maintain most of the already existing code, making available the Grid potentials to WebSPN. Thus, we have exploited the Grid authentication and security mechanisms to manage the corresponding aspects in our application, and the other services to implement a more efficient load distribution policy.

We have also introduced a fault detection and recovery algorithm to prevent job faults.

3.1 Job distribution: the Workload Scheduler

![Figure 2. Workload Scheduler Algorithm Schematization.](image-url)
The Workload Scheduler manages the jobs distribution over the computational Grid. Before distributing the jobs, it is necessary to determine which are the available machines. The distribution will also take into account the resources availability of each machine. The algorithm, pictorially depicted in Fig. 2, is schematized through the following sequence of steps:

1. **Grid Machines List Configuration** - The first phase consists in the identification of the machines that physically compose the computational Grid: the user has to create a file that reports these machines according to the FQDN (Fully Qualified Domain Name) notation.

2. **User Proxy Credentials Creation** - Temporary user proxy credentials have derogated from the user credentials to authenticate the Grid machines, exploiting the GSI protocol. Thanks to the delegation mechanism it is possible to access all the resources or services by only one authentication.

3. **Active Machines Verification** - After the creation of the user proxy credentials a diagnostic verification of the Grid Resource Allocation Manager (GRAM) and the Monitoring and Discovery Service (MDS) is performed. In case of the GRAM, the MASTER asks each machine that compose the Grid cluster to verify the activation of the globus-gatekeeper daemon in the default port 2119, exploiting the globus.gatekeeper_client.ping service. It also controls that the job manager service provided by the globus-gatekeeper is the jobmanager-fork. After a positive GRAM test the MDS services are controlled. On each machine of the cluster a GRIS server of type OpenLDA must be active. It is launched as a SXXgris daemon, that processes the requests about the resources exploitation, coming from the MASTER. To connect to a remote LDAP server a client must authenticate itself through the related Grid Security Infrastructure (GSI) service.

At the end of this phase the active machines list is compiled by dropping from the Grid machines list the machines that do not satisfy the previous requirements.

4. **Resources Availability Calculation** - Each active machine communicates its available resources to the MASTER every 30 seconds, through the LDAP server, as specified in its grid-info-resource-ldif.conf file. The information of node $i$ are: $MEM_{tot}(i)$ (the total amount of RAM Memory - Mds-Memory-Ram-Total-sizeMB) $MEM_{free}(i)$ (the percentage of RAM Memory free or available - Mds-Memory-Ram-Total-freeMB) $MEM_{cache}(i)$ (the second level cache dimension index - Mds-Cpu-Cache-l2kB) $CPU_{tot}(i)$ (an index that quantifies the CPU performances - clock frequency Mds-Cpu-speedMHz, number of processors Mds-Cpu-Total-count, type of processor Mds-Cpu-type) $CPU_{free}(i)$ (the available CPU percentage since the last 5 minutes - Mds-Cpu-Free-5minX100)

The parameters values related to all the active machines are collected by the MASTER into a separate set for each parameter. All these indices, excluding the percentages, are normalized in the $[0, 1]$ range by rating to the maximum value in the set. After this elaboration they are synthesized in the $MDS(i)$ parameter:

$$MDS(i) = \lambda_1 MEM_{tot}(i) MEM_{free}(i) + \lambda_2 MEM_{cache}(i) + \lambda_3 CPU_{tot}(i) CPU_{free}(i)$$

In (4), $\lambda_1$, $\lambda_2$ and $\lambda_3$ are constants that weight the related components such that: $\lambda_1 + \lambda_2 + \lambda_3 = 1$. From preliminary experiments, a good behavior has been obtained through the following settings: $0.6 \leq \lambda_1 \leq 0.9$, $0.01 \leq \lambda_2 \leq 0.1$ and $0.1 \leq \lambda_3 \leq 0.35$. The higher value of $\lambda_1$ points out that the application behavior is strongly affected by memory requirements.

5. **Inducted Load Estimation $T$** - An approximation for the workload generated from the application is computed by using the upper bound $L$ of equation (2) as follow:

$$T = \frac{L \cdot sizeof(Arcs)}{N}$$

$T$ represents a pessimistic estimation of the workload equally distributed to each job, that will be used in the load distribution algorithm.

6. **Load Distribution** - The distribution of the workload in the Grid cluster is performed considering the resources availability, periodically estimated by the MDS service. These information are elaborated through a function cost that optimizes the distribution, univocally identifying the list where the state to be processed has to be sent:

$$M(MDS, DPHM_a, N, T)$$

The states to be processed are equally distributed by the hash function of the equation (1) into $Q = 10N$ sets $SOS_k$, where $1 \leq k \leq 10N$ identifies the set.

The distribution of the sets over the Grid cluster is performed distinguishing the two cases related to the satisfaction of the condition $T < Min(MDS)$: if verified all the machines have enough resources to schedule the same number of jobs, 10 for each; in the other case the MDS values must be taken into account. In the case of identical and unrelated machines, the distribution algorithms is formalized in the following equation:

$$N_i = 10N \frac{f(N, MDS(i))}{\sum_j f(N, MDS(j))}$$

where $N_i$ is the number of the sets associated to machine $i$,

$$f(N, MDS(j)) = N_i^{MDS(j)}$$
and

\[ MDS_{opt}(j) = \lambda_1 MEM_{tot}(j) + \lambda_2 MEM_{cache}(j) + \lambda_3 CPU_{tot}(j) \]

The set of states associated to the generic machine \( i \), \( \mathcal{JS}_i \), is obtained by joining the \( SOS_p \) sets such that:

\[
\mathcal{JS}_i = \begin{cases} 
\{ \bigcup_p SOS_p \in SOS \mid 1 \leq p \leq N_1 \} & i = 0 \\
\{ \bigcup_{k=1}^{i-1} SOS_p \in SOS \mid \sum_{k=1}^{i-1} N_k < p \leq \sum_{k=1}^{i} N_k \} & i > 1
\end{cases}
\]

Equation (5) is derived from the ASSIGN-U algorithm defined in [11], where it is also demonstrated the algorithm competitiveness for uncorrelated machines with permanent jobs.

7. Application Execution - The application is distributed into the \( N \) Grid machines by generating \( N \) sub-jobs through the jobmanager-fork invocation, one for each machine. For each sub-job, a request to the GRAM is generated; the GRAM authenticates the user and then generates the temporary credentials necessary to initialize the computation.

MPICH-G2 exploits the Globus IO services over TCP/IP to implement the communication between the sub-jobs. MPICH communicates with GTK-2.4 through RSL instructions interpreted by the DUROC, that also synchronizes the sub-jobs execution. At the beginning, the sub-jobs are synchronized by the MPI_Init(...) invocation. At the end of the computation, each sub-job sends the results to the MASTER coordinator to reorganize them, and terminates its execution by MPI_Finalize(...).

3.2 Fault Tolerance

In order to preserve the GridSPN elaboration from eventual faults, it is defined a Fault Recovery System (FRS) to detect and recovery fault conditions. The FRS is activated in the MASTER machine as a thread and performs its task by iterating a polling cycle until the application terminates. Fig. 3 shows a logic schematization of the FRS algorithm described in the following. Faults are classified into three classes: Machine Fault, Link Fault and Job Fault. A Machine Fault (MF) is a critical event that abruptly interrupts the machine elaboration, as unexpected blackout, bus crash, devices failures, shut-down, ...: the Link Fault (LF) is specifically referred to the network conditions: link, hub or firewall breaches, TCP-IP protocols error; the Job Fault (JF) identifies premature job killing or termination conditions, like, for example: explicit killing from another user, error in request format, problem in the resources management, reboot.

Fault Detection

As shown in Fig. 3, the algorithm is organized into two phases: in the first the MASTER performs an Active Machines Verification as in the step 3 of the Workload Scheduler algorithm (subsection 3.1). Once this task is performed, the job status is verified (second phase) using the GT2 pid that univocally identifies the job associated to the specific machine in the Grid cluster. Each asked machine will return the job status: pending if it is waiting for resources allocation, active if it is running, done when it completes its elaboration or failed if an error was occurred.

Then, if a machine experienced a fault or a job failed, it is excluded from the Active Machines List and it is signaled to the FRS to recover the data and to restore the elaboration. The FD algorithm terminates when the elaboration returns the analysis results and all the resources are released.

Fault Recovery

Once the fault has been detected, the FRS will recover the data and computation lost due to the job fault. To this purpose it is defined an array of \( N \) bytes, \( MS \), that describes the state of each machine in real time, and an integer variable \( F \) to count the number of faults, initialized to 0 (of course \( F < N \)). Each machine has a local copy of \( MS \) and \( F \), instantiated in runtime and managed from the FRS. If the Fault Detection Process notifies a fault condition in machine \( i \), the MASTER first deletes this machine from the Active Machines List, then set \( MS[i] \) to 0, increments \( F \) and finally updates the local copy of these variables through a broadcast message. Now the \( N \) sets assigned to machine \( i \), \( \mathcal{JS}_i \), must be redistributed over the active machines. This operation requires to modify the distribution algorithm, as reported in Fig. 4. Once the computation of set \( (k) \) and machine \( (i) \) to which an expanded mark will be sent is completed, the availability of the selected machine is controlled by consulting the \( MS \) array. In the fault case \( MS[i]=0 \), the
state is redistributed to another machine \( t \), established by the equation:

\[
t = \begin{cases} 
  t_1 = (k - \sum_{j=1}^{j-1} N_j) \mod P & \text{if } t_1 \neq k \\
  t_2 = (Ni \mod P + 1) \mod P & \text{if } t_2 \neq t_1 = k \\
  t_3 = (Ni \mod P + 2) \mod P & \text{if } t_1 = t_2 = k 
\end{cases}
\]

(6)

where \( P = N - F \).

When the distribution is reorganized after a fault, the FRS process restarts from the initial marking \( M_0 \) to rebuild the cache of the expanded markings lost in the fault. From a performance point of view, the impact of this solution is reduced thanks to the expansion algorithm: only the lost markings are expanded, the others are saved in the caches of the corresponding machines.

4 Conclusions and future works

This paper describes the porting of the WebSPN tool to a Grid infrastructure, i.e. how to transport the existing MPICH implementation into the GT2.4 environment. The reason that motivated the choice of GT2.4 is MPICH-G2: the latest grid-enabled interface of the MPICH protocol. It is foreseen, as future work, the development of a GridSPN Grid service over the GT3 infrastructure.

The services offered by the Grid platform (MDS, GRAM, GRIS, GIS, DUROC) permitted to introduce some new interesting features, like a robust security management, an adaptive Workload Scheduler and a Fault Recovery System. Further enhancements to these algorithms are in progress. Current testing phase will soon exploit in a complete case study, to evaluate the behavior of the new implementation.

References