Time Warp Simulation of Stochastic Petri Nets

Hany H. Ammar

Dept. of Electrical &
Computer Engineering
West Virginia University
Morgantown, WV 26506

Su Deng

Dept. of Electrical &
Computer Engineering
Clarkson University
Potsdam, NY 13676

Abstract

This paper addresses the problem of developing parallel simulation techniques to analyze complex Stochastic Petri Net (SPN) models. The approach of parallel simulation is to divide a general SPN spatially into several connected subnets. The various subnetworks are simulated in parallel by several logical processes (LPs) which synchronize. The rich and complex structure of Petri Nets necessitates the development of an algorithm which can handle general forms of network partitions.

In this paper, an algorithm based on the Time Warp strategy for optimistic parallel simulation is presented. Time scale decomposition is also used with spatial decomposition to induce parallelism and reduce synchronization overhead. An example of performability analysis using both spatial and time-scale decomposition is presented.

1 Introduction

Discrete-event simulation of complex systems is one important application in which parallel processing techniques have been applied with noticeable success. These techniques are expected to improve our ability to efficiently simulate the behavior of large scale systems over a long period of time. They are also expected to reduce the response time of interactive simulations used for training purposes in many applications.

A good survey of the literature on parallel simulation techniques, mainly based on spatial decomposition, has been reported by Kaudel [2]. The various techniques can be classified into two categories according to the type of synchronization between LPs. These are the conservative techniques [5], and the optimistic techniques [1]. The conservative simulation prevents out-of-order event processing by forcing an LP to block whenever it is possible to receive a message with an earlier timestamp than the earliest message in its local event list. In optimistic simulations, LPs synchronize by state savings and rollbacks processes events.

This paper addresses the problem of developing parallel simulation techniques to analyze Stochastic Petri Net (SPN) models. The approach of parallel simulation is to divide a general SPN spatially into several connected subnets. The rich and complex structure of Petri Nets necessitates the development of an algorithm which can handle general forms of network partitions. Although parallel simulation of queueing networks has been extensively studied [3], SPNs, to the best of the authors knowledge have not been considered. We are only aware of the work in progress by Thomas [4] on parallel simulation of Petri Nets. The protocol developed in [4] is based on the conservative approach. Moreover, each place and each transition in the net is represented by a logical process. This protocol will be difficult to use with large scale networks consisting of many places and transitions.

In this paper, an algorithm based on the Time Warp strategy for optimistic parallel simulation [1] is presented. The various subnetworks are simulated in parallel by several logical processes (LPs) which synchronize by rollbacks. Each logical process (LP) simulates a subnetwork and advances its local simulation time (i.e., the accumulated firing time) as far as it could. When a token is needed by another subnetwork, a message with the local simulation time (called token time) will be sent to the proper LP. When a simulation error is detected in a LP, such as receiving a message with a small token time (this indicates that the previous simulation did not consider the effect of the incoming token) the LP will roll back to the simulation time indicated by the token time of the received message and re-simulate the firing process from that point on.

TH0386-3/91/0000/0186/$01.00 © 1991 IEEE

Time scale decomposition can be used to exploit parallelism and reduce overhead. It was shown in [10] that massive parallelism can be achieved in the simulation of a hierarchical system by dividing the simulation into several phases which corresponds to simulating one level of hierarchy at a time using a bottom-up approach. At the lowest level of the hierarchy, a logical process can be created to simulate a given component for a certain state of the next higher level. In this case a potentially large number of independent LPs can be used to simulate a given component for all possible higher level states. These higher level states will be reached at different times as the system evolves. Therefore, in essence the behavior of a system component at is simulated at different instances of the simulation time in parallel. Moreover, the created LPs do not need to synchronize. Logical processes simulating different components at the same higher level state will be dependent and will need to synchronize. At the higher level, the simulation of the macro components will make use of the simulation statistics obtained at the lower level.

In following section we present some definitions and notations. In section 3 an algorithm for parallel simulation of SPNs based on spatial decomposition is described, and in section 4 a performability analysis example is presented to illustrate the use of time scale decomposition and spatial decomposition in the simulation of SPN models.

2 Definitions and Notations

The approach of parallel simulation is to divide a general SPN into several subnets. Each subnet i is simulated by a logical process (LP_i). Transition firings in the subnets are simulated in parallel by the LP's. Tokens moved among subnets are represented by messages transferred among LP's.

It is assumed in the following discussion that an SPN is defined as follows: $SPN = (P, T, A, R)$ with an initial marking M_1 , where $P = \{p_1, p_2, ..., p_n\}$ is a set of places, $T = \{t_1, t_2, ..., t_m\}$ is a set of immediate and timed transitions, $A \subset \{P \times T\} \cup \{T \times P\}$ is a set of arcs including inhibitor arcs, and $R = \{r_1, r_2, ..., r_k\}$ is a set of firing rates of timed transitions. Moreover, it is assumed that a probability distribution is specified for the firing of immediate transitions forming a random switch. The transitions firing rates and firing probability can be marking dependent, i.e., they depend on the current marking of the SPN.

Definition 2.1 *An SPN partition.* An SPN partition is a set of N subnetworks such that, $SN_i = \{P_i, T_i, A_i, R_i\}, i = 1, \cdots, N$, where $\cup_{i=1}^{N} P_i = P$, $\cup_{i=1}^{N} T_i = T$, the set $A_i \subset \{P_i \times T_i\} \cup \{T_i \times P_i\}$, and The set R_i is the set of rates of local transitions in T_i.

Definition 2.2 *The input and output places and transitions of a subnet.* A place $p_i \in P_i$ in subnetwork i (SN_i) is said to be a member of the set of input places (IP_i) of the SN_i if there exists a transition $t_j \notin T_i$ with p_i as an output place, i.e, $\{t_j, p_i\} \in A$. Similarly, a place $p_k \in P_i$ in SN_i is said to be a member of the set of output places (OP_i) of the SN_i if there exists a transition $t_j \notin T_i$ with p_k as an input place, i.e., $\{p_k, t_j\} \in A$. In general, it may be possible that $p_k \in IP_i \cap OP_i$. Input and Output transitions of a subnet are defined in exactly the same manner if we interchange places by transitions.

In Figure 1 (a), p_1 is an input place, and t_1 is an output transition for SN_1, t_2 is an input transition of SN_2, and p_5 is both an input and an output place of SN_3.

Definition 2.3 *Global arcs.* A global arc is an arc connecting a place (transition) in SN_i to a transition (place) in SN_j.

Global arcs can be divided into three types: place-transition arcs (PT arcs), transition-place arcs (TP arcs), and inhibitor arcs (IN arcs). In Figure 1 (a), $\{t_1, p_3\}$ is a TP arc, $\{p_5, t_2\}$ is a PT arc, and $\{p_4, t_2\}$ is an IN arc.

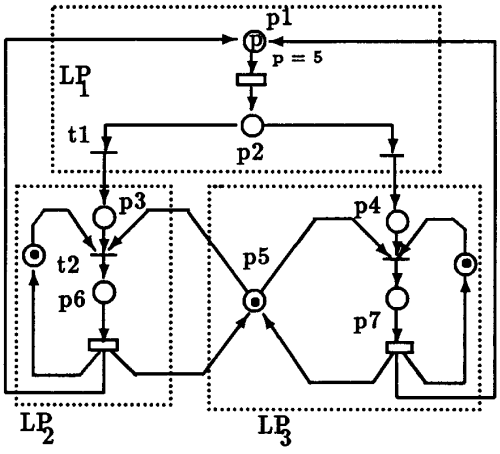
When separating subnetworks, the global arcs are removed and local places and arcs are added to each subnetwork. These added places become images of input/output places in other subnetworks. For example, when separating subnetworks as shown in Figure 1 (b), the TP arc $\{t_1, p_3\}$ is removed and a local place np_1 and an arc $\{t_1, np_1\}$ were added to SN_1. Similarly, places np_2, np_3, and np_4 were added to SN_2 and the corresponding global arcs were removed.

The place np_1 is an image of place p_3 in SN_2. Similarly places np_2, np_3 and np_4 in SN_2 are images of places p_4, p_5 and p_7 in SN_3, respectively. In this case messages are sent between subnets to ensure that the marking in a place in one subnet is consistent with the marking in its image in the other subnets.

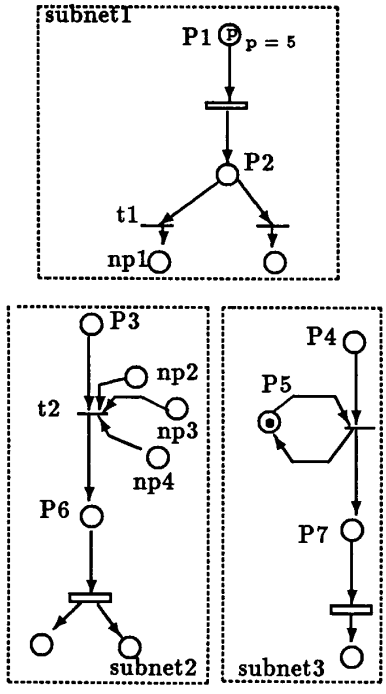
The interactions between subnets in the form of messages are classified into 5 types of messages as follows:

a token message (TM). this message carries a token passing through a TP arc. In this case a token is taken from an added place np_i and sent to its image.

a token request message (TRM). This message is used to simulate a transition firing action through a PT arc. In this case, an



(a)



(b)

Figure 1: SPN partition

input transition is requesting a token from a place in another subnet.

an inhibitor message (INHM). This message carries the inhibiting condition specified by an inhibiting arc.

a token request acknowledgment (TR-Mack). This message is used as a response to a TRM or an INHM.

cancel token message This message is used in the rollback mechanism needed for synchronization.

Next we define the time-stamp values associated with tokens transferred between subnetworks.

Definition 2.4 *Token Time* (TT). A token time, $TT(a,k)$, is a time-stamped value for token a represented by a message sent from SN_i to place k in SN_j. The value is determined by the sending LP, i.e., LP_i.

A token time (TT) has different meanings in the different types of messages. A TT in a TM or a TR-Mack from LP_i represents the accumulated firing time in LP_i. The TT in a TRM or an INHM received by LP_i from LP_j represents the next local clock value of LP_j if and only if LP_j can get its required tokens from LP_i and fire the corresponding transition. TT in a CTM is used to indicate which message should be canceled due to a rollback in the local simulation.

We finally define the local clock and its dependence on the TT of input messages and transition firing.

Definition 2.5 *Local clock* : A local clock of SN_j, denoted as LC_j, is an accumulated value of firing time. It is advanced from $LC_j^{t_l-}$ to $LC_j^{t_l+}$ after transition t_l fires ($t_l \in T_j$) with a firing time FT_l as follows :

$$LC_j^{t_l+} = \max_k\{TT_k, LC_j^{t_l-}\}+FT_l, TT_k = \min_a\{TT(a,k)\}$$

Here, TT_k is the minimum token time of all tokens in place k, and For all p_k such that $\{p_k, t_l\} \in A_j$.

Each LP selects a transition t_l to be fired next from its local enabled transitions under the current local marking. After t_l fires, the local clock is advanced to LC^{t_l+} and a new local marking is created. At this time, some tokens in output places may need to be sent to other LP's. Similarly, some tokens may arrive from other LP's. Before the next firing, LP will process its input/output messages and update its local marking.

3 The Algorithm

The main task of our algorithm is to synchronize the LP's simulating the various subnetworks, and to implement the functions of the global arcs between

subnets. In the following algorithm we assume that the firing rate or probability of a transition in SN_i depends only the local markings of the subnetwork.

3.1 LP's synchronization

Synchronization among LP's is controlled by the token time (TT) of a message, the local clock and the CTM messages. As in the Time Warp mechanism, each LP uses an input queue for all received messages. The messages in the queue are ordered by TT. The message with a smallest TT is at the head of the queue. A LP will process these messages by checking the head message as follows:

> if (TT of the first message < clock), simulation rolls back to the time where the clock <= TT, input and process all messages with their token times equal to TT.
>
> if (TT = clock) or (TT > clock and no local firing is possible), input and process all messages with the same TT.
>
> if (TT > clock) and (a local firing is possible), do not input message, continue local firing process.

The above conditions will ensure that a LP only receives a message when its clock reaches to the TT of the message; therefore, a message received earlier with a large value of TT will wait in the queue; a later received message with a small value of TT will let the LP roll back to re-process the message. When a roll-back is necessary, the LP will set back its clock to the simulation time just before the TT to re-simulate firing process and consider the token presented by the message. The LP will also cancel all messages generated before the roll back by sending cancel token messages (CTMs).

When a LP receives a CTM, and the token to be canceled have not been processed, the LP simply deletes the token in its input queue. However, if the token has been processed, the LP should roll back to the state at the time indicated by the cancel token message.

3.2 Global arcs simulation

The functions of the global arcs are simulated by different types of messages as mentioned above. For a TP arc when its transition fires, a TM message is used to update the marking in the target place.

For a PT arc, we only need to consider the difficult case in which the target transition t_j has more than one input place in different LP's. In this case, whether the transition t_j can be enabled will depend on the marking in all of its input places. In parallel simulation, since a LP only has the information on its local marking, it has to obtain information from other LP's for the firing of t_j by transferring messages.

In our algorithm, TRM and TRMack are used for simulating PT arcs as follows. Consider a transition t_j in SN_j with input places in SN_j and SN_i. LP_j will check first the local enabling conditions. When all the local enabling conditions are satisfied, LP_j will assume that t_j is enable and determine the next transition to be fired among all local enabled transitions. If t_j is the next transition to be fired, LP_j will send a token request message (TRM) to LP_i to ask for tokens or for inhibiting conditions when the PT arc is an inhibitor arc. As in the Time Warp, it is not necessary to wait for the response. The local simulation in LP_j can be continued under the current local marking, i.e., by firing the next enabled transition.

A TRM message sent from LP_j to LP_i describes some basic information of LP_j, including *token time* to indicate the next local clock value $LC_j^{t_j+}$ if t_j will be fired, an input place list(or a p_k *list* to indicate which place in SN_i is related to t_j and what kind of relation exsits (i.e., whether an input place is an inhibit place or not). When LP_i inputs and processes the TRM, it checks all places indicated by the p_k list by the following two conditions (called *positive response conditions*): (1) if p_k is an input place of t_j and there are tokens there; (2) if p_k is an inhibiting place of t_j and there are no tokens there. When the positive response conditions are satisfied in all input places, LP_i will send a TRMack with an answer with a positive response to LP_j, then updates its local marking, if TRMack contains tokens sent to LP_j according to its request. When a place in the p_k list does not satisfy the positive response conditions, LP_i will hold the TRM and continue its local simulation until the conditions are satisfied, then send a TRMack with a negative response together with its clock value as TT to indicate when the conditions can be satisfied.

When LP_j gets all response messages from other LP's, it will check the answers of the TRMack messages. If all TRMack messages are positive, t_j will be fired in SN_j as assumed before. If there is any TRMack with a negative response, t_j is not ready to fire before the time indicated by the maximum TT of the TRMack messages with the negative answer. In this case LP_j has to return all tokens carried by the positive TRMack messages by sending back TM messages to the senders. At the time TT in SN_j, t_j may or may

not be enabled according to local marking conditions. If it is still enabled, new TRM messages at TT will sent and the above process is repeated.

3.3 The Algorithm

The algorithm can be briefly described as follows:

1. Check input queue :
   if (input queue = empty), goto 4.
2. Input messages :
   If (message type = CTM), cancel the message indicated by CTM and corresponding output messages, goto 1.
   If (TT > clock and local firing is not possible) or (TT = clock), input all messages with the same TT, goto 3.
   If (TT < clock), LP rolls back to clock <= TT, input all messages with the same TT, goto 3.
   Goto 4.
3. Reset local marking by checking each inputed message :
   Set $StateSaveFlag$ = FALSE, set clock = TT.
   Repeat the following operations for each inputed message :

   In the case of TM :
   Change local marking, set $StateSaveFlag$ = TRUE,
   In the case of TRM : check the required place in the p_k list,
   If (response conditions are satisfied for all places in the p_k list), send TRMack(clock,YES) to the required LP, change local marking, set $StateSaveFlag$ = TRUE.
   If (response conditions are not satisfied), set $TRMwaiting$ = TRUE.
   In the case of TRMack :
   If (TRMack answers YES), set $StateSaveFlag$ = TRUE, change local marking.
   Increase $messageINcounter$ by one.

   If ($StateSaveFlag$ = TRUE), save the current state, including local clock, local marking, some flags for LP rollback.
4. Ask tokens or state from other LPs :
   If (next fired transition t_j needs tokens from other LPs) and ($messageOUTcounter$ = 0), calculate next clock value LC^{t_j+}, send TRM with TT = LC^{t_j+} to other LPs, set $messageOUTcounter$ = number of outgoing TRM messages, goto 1.
5. One step local firing :
   If ($messageINcounter$ = $messageOUTcounter$ > 0) and (the transition that gets tokens from other LPs is NOT enable), return all required tokens from other LPs by sending TM messages, save the current states, set $messageINcounter$ = $messageOUTcounter$ = 0.
   Step firing under the current marking, advance clock, send output tokens, save the current state.
6. Process all waiting TRM messages :
   If ($TRMwaiting$ = TRUE) and (the response conditions of the TRM are satisfied), send TRMack(clock,NO) to the required LP, reset the $TRMwaiting$.
   Goto 1 until simulation finishes.

A correct simulation sequence is maintained by the rollback conditions and conditions in step 2 specified for all input places in a subnet. It is clear that for each subnet, if all input places receive tokens in a correct simulation order, the simulation on the subnet should be correct when the tokens are only used in local transitions. When a token enables several transitions that are in different LPs, the algorithm will select a correct transition to be fired by exchanging TRM and TRMack messages between LPs.

When a LP needs to roll back, it may send cancel token messages to other LPs. It is not possible for all LPs to be involved in the rollback forever. The reason can be explained as follows. Assume that LP_i needs to roll back to time t_1. Any canceled token message sent to LP_j by LP_i must have token time t_2, where $t_2 >= t_1$, because the message must be processed after t_1 in LP_i. Therefore, for secondary rollback, LP_j may roll back to t_2 and may send its cancel token message with token time t_3, $t_3 >= t_2$. Finally, at the kth rollback, t_{k+1} will increase to the time point that has not been simulated, and the rollback process stops. Thus, simulation can be started from the rollback points with a correct order.

4 An Example of Performability analysis of a Fault-Tolerant Distributed system

In this section, an example using SPNs for performability analysis of a distributed system is presented. The performability model consists of a performance model, and a reliability model. The former describes the computation, and synchronization activities in the application software, while the later, also called the component failure and repair model, defines the current configuration of processors or hardware resources and modules or software resources available for the computation. These two models define the reward model needed for performability analysis. Due to the limitation in space, we will concentrate only on the performance model in this section.

4.1 Example description

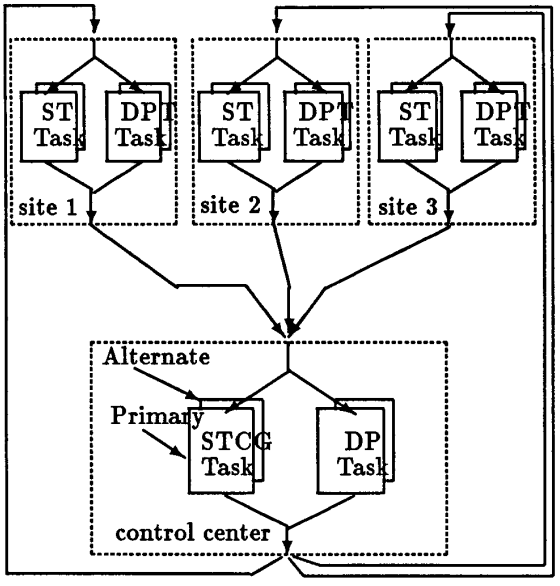


Figure 2: Parallel task structure of a distributed system

Figure 2 shows an example of a parallel task structure in a distributed system. The system consists of 4 nodes: 3 independent sites and a control center. Each site is connected through a highly reliable high speed communication channel to the control center. There are two parallel task in each node. The ST task and DPT task in a site (the STCG task and DP task in the control center) are cooperating concurrent modules. During execution, they exchange data and synchronize at the end to complete the process. After synchronization, a site sends its data to the control center and waits for the results to come back.

In this example, the *Recovery Block* technique is used for software fault tolerance [8] [9]. Each task has one primary and one alternate module. The primary module in the recovery block uses the most efficient algorithm and is intended to be used in the normal operation of the software. The alternate module performs the desired operation of the primary module in a different manner, and generally uses less efficient but simpler (hence more robust) algorithm. When a primary module fails, its alternate module is used to keep the system running during the repair period of the primary module.

In the event of a failure in one of the primary modules, its related computation has to roll back to the nearest recovery point, then the computation is restarted from the recovery point because of the interprocess dependencies in cooperating concurrent modules. For example, when a ST task fails, its related module (the DPT task) will also roll back to their recovery point with the ST task to re-start computation. In our example, we assume that the related parallel tasks has only one recovery point — that is in the event of a failure the related tasks will repeat all computations done in the interval from forking to failure.

A performance model for the system in Figure 2 is shown in Figure 3 using the SPN representation. Tokens in place p_1 and p_2 represent the ST task and the DPT task running with computation time t_1 and t_2 respectively in site 1. Transition t_3 represents a failure event of the ST task (in site 1) during the computation (if there is a token in p_1). Since we have only one recovery point, in the event of a failure, the ST task will roll back to the beginning with its cooperating module (the DPT task). The rollback process is implemented by the immediate transitions when a failure happens. The time required by the rollback operation is represented by transition t_5. Transition t_6 represents the necessary synchronization of the ST task and the DPT task, and transition t_4 represents the failure event during synchronization. It is clear that t_4 indicates that the failure can only happen during the task synchronization but not while waiting for it, i.e., a failure can not happen when only one token is in either place p_8 or p_9. A token in p_7 represents a state where the control center is waiting for data from a site and available to process the data immediately. A token in p_{17} repre-

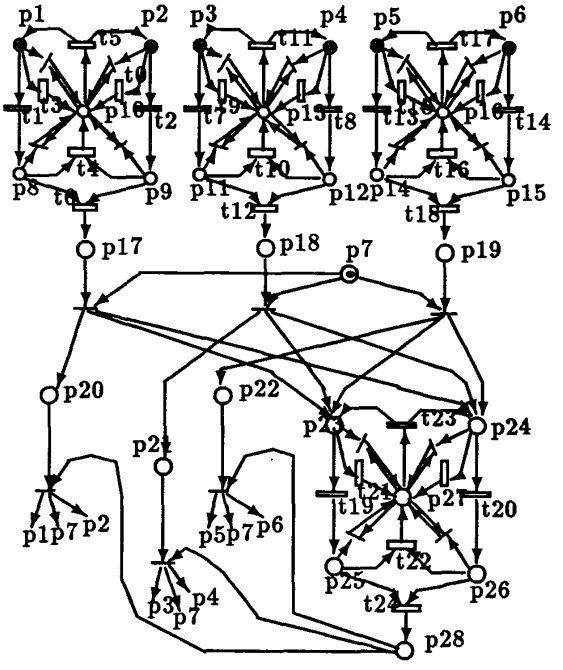


Figure 3: SPN representation of the performance model

sents a state where the data from site 1 is waiting to be processed by the control center. This model describes the behavior of the system shown in Figure 2, including parallel task executions, synchronization, primary module failures, rollback operations, data waiting and processing in each site. Several performance measures can be derived from the performance model, such as average waiting time of a site, utilization of a primary module, the mean response time of the control center, and so on.

4.2 Time scale decomposition and spatial decomposition

In this section the decomposition of the above model for parallel simulation using both spatial decomposition (SPD) and time scale decomposition (TSD) will be described.

It was shown in [10] that when a model contains fast and slow activities, SPD will suffer from a large number of rollbacks. The reason for such a behavior is the large asymmetry existing between different processes simulating activities in different time scales. This asymmetry limits the parallelism in the simulation and increases the synchronization overhead. It was shown that the performance is greatly enhanced when TSD is applied first. This is because TSD increases parallelism and reduces overhead in the simulation process.

Time scale decomposition

In the following paragraphs the time scale decomposition of the network in Figure 2 is described.

Time scale decomposition divides a large network into small subnets by separating fast activities and slow activities into different subnets. In our example in Figure 2, the parallel task executions are fast activities. The failure of a primary module and related rollback processes are slow activities.

There are two basic steps in TSD: identifying fast subnets, and aggregating them into a slow time scale subnet. A formal representation of TSD is given in [7].

In the first step, we only consider the activities at the fast time scale. In this case, we assume that the slow events will never happen. Therefore, when the places and transitions related to the slow events are removed, the whole SPN will decompose into fast subnets. In Figure 3, place p_{10}, p_{13}, p_{16}, p_{17} and the transitions connected to these places should be removed because they represent slow activities (the failure of a primary module and the rollback processes). Figure 4 shows the fast subnet of the SPN in Figure 3 after the places and the transitions have been removed. This fast subnet describes the short-run dynamics of the system for a given local initial marking.

In the second step, we consider the activities in the slow time scale. In this case, the fast subnet is aggregated into a single place with the other places removed at the first step. The transitions in the slow subnet are equivalent to the transitions connected between the fast subnet and the removed places in the original SPN.

Figure 5 shows the slow time scale subnet. It describes the rollback processes when a primary module fails in the system. Place p_1 is the aggregated place of the fast subnet. The other places in the figure are the same as that in Figure 3. The number of tokens in the aggregated place is equal to the number of tokens moving out of the fast subnet. According to the model shown in Figure 3, the failure and rollback processes can only happen in at most 3 places at the same time; therefore, there are only 3 tokens in place p_1.

Transition t_5, t_{11}, t_{17} and t_{23} in Figure 5 are the same as that in Figure 3. However, the transition rates of t_1, t_2, t_3 and t_4 in the slow time scale subnet are dependent of the fast subnet due to the ag-

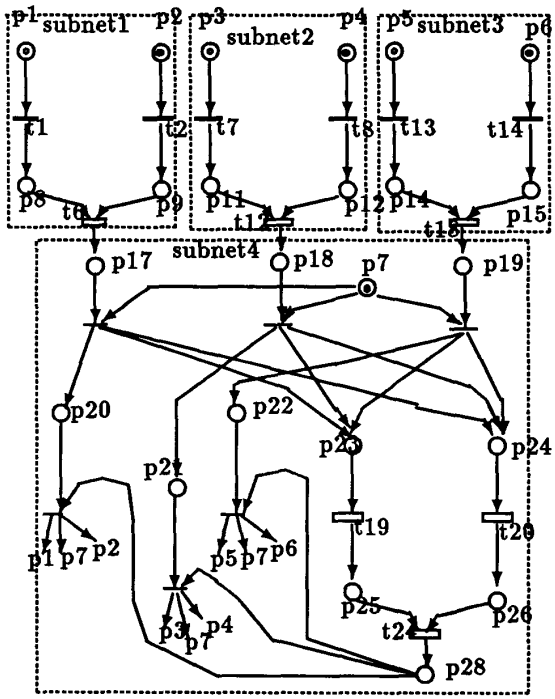


Figure 4: Fast time scale subnet of the performance model

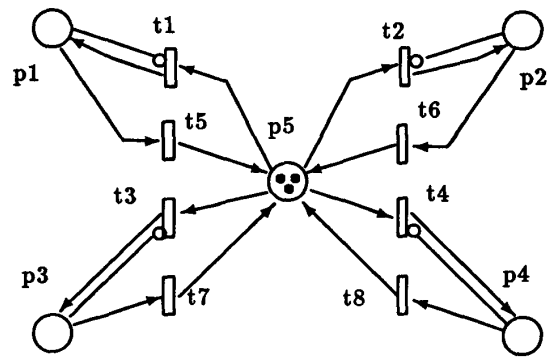


Figure 5: Slow time scale subnet of the performance model

gregation. We can decide these transition rates from the token distribution of the fast subnet and the related transition rates in the whole SPN in Figure 3. For example, the rate of transition t_1, defined as r_1, can be derived from the following equation:

$$r_1 = P(p_1 = 1) \times r_3 + P(p_2 = 1) \times r_0 + P(p_8 = 1 \text{ and } p_9 = 1) \times r_4$$

Here, $P(p_i = 1)$ is the steady-state probability that there is one token in place p_i in the fast subnet. $P(p_8 = 1$ and $p_9 = 1)$ is the steady-state probability that there is one token in both place p_8 and place p_9 in the fast subnet. r_i in the equation is the rate of the transition t_i in the whole SPN shown in Figure 3. The product $P(p_1 = 1) \times r_3$ indicates that a failure may happen (i.e., transition t_3 fires) with a rate r_3 if and only if the primary module of the ST task is running (i.e., there is a token in place p_1). The other products in the equation correspond to other failure cases for the failure of the primary module of the DPT task and the failure of the task synchronization. Thus, t_1 in the slow time scale subnet represents all possible failure cases of the primary modules in site 1. In general, the product for the failure associated with place p_i, p_j, $\cdots$, p_l can be expressed by $\sum_{m=0}^{n}(r_{km} \times d_m)$, where r_{km} is the transition rate of t_k under marking m associated with place p_i, p_j, $\cdots$, p_l in the whole SPN; d_m is the joint probability under marking m in the fast time scale subnet; n is the total number of markings that enable transition t_k when place p_i, p_j, $\cdots$, p_l have tokens. The rates of transition t_2, t_3 and t_4 can be decided similarly.

Spatial decomposition

SPD is applied after using TSD in order to further induce parallelism and simulate a fast subnetwork by several LPs. As in the example shown in Figure 4, the whole net is spatially divided into four subnets indicated by the doted lines. The subnets are simulated by four LPs in parallel using the SPD algorithm presented in section III.

5 Results and Conclusions

The SPD algorithm has been implemented on a MULTIMAX 18 processor system. The results of two experiments are described in this section along with some concluding remarks.

In the first experiment, the example shown in Figure 1 was used to check the correctness of the algorithm. As shown in the figure, the SPN was divided into three subnets; therefore, three LPs were used to simulate each subnet. After the net partition, all kinds

Table 1: Results from the first experiment

Number of tokens	Probability in place P_1 parallel simulation	analytical solution	% error	precision
0	0.31797	0.29516	7.7%	2.9%
1	0.22114	0.22527	-1.8%	1.7%
2	0.21571	0.22401	-3.7%	1.7%
3	0.15324	0.16163	-5.2%	2%
4	0.07325	0.07412	-1.2%	3.8%
5	0.01949	0.01982	-1.6%	6.7%
Number of tokens	Probability in place P_3 parallel simulation	analytical solution	% error	precision
0	0.30529	0.2988	2.2%	4.1%
1	0.37921	0.3742	1.3%	2.5%
2	0.24131	0.2475	-2.5%	4.0%
3	0.06764	0.0727	-6.9%	10.1%
4	0.00655	0.0066	-0.8%	16.2%
5	0	0	0	0
Number of tokens	Probability in place P_5 parallel simulation	analytical solution	% error	precision
0	0.62004	0.621504	-0.2%	1.3%
1	0.37996	0.378496	0.4%	2.1%

of the global arcs existed and should be simulated by message exchanging among the LPs. During the experiment, LP's synchronization and subnet arcs simulation were examined by checking the local firing, clock advance, simulation sequence as well as rollback processing and dead lock.

The simulation results (token distribution in a SPN) were compared with the results obtained from analytical solutions using the package *GreatSPN* [6]. Table 1 shows the comparisons, including estimated values, relative errors and precisions of confidence intervals. The parallel simulation results were obtained from 23 independent runs for the estimated values and 95% confidence intervals. The precision in the table indicates the relative interval width around the estimated point.

The second experiment used the performance model shown in Figure 4 for performability analysis. The model was divided into 4 subnets as shown in the figure. Each subnet was simulated by a LP. In this partition, only one type of global arc (TP arc) existed among the subnets; therefore, the simulation process in each LP was much simpler than that in the first experiment. According to the algorithm, when only TP arcs existed, an LP simply sent TM messages. It did not need to create TRM, INHM messages and to wait for their TRMack messages, which reduced the number of messages created and transferred among LPs. In this case, the overhead of the algorithm is the same as the Time Warp. However, when more types of global arcs exist among the subnets, more overhead might be involved with the changes of the partition.

Table 2: Results from the second experiment

place and tokens	parallel simulation	analytical solution	% error	precision
p_1 : 0	0.545052	0.547811	-0.5%	1.2%
1	0.454664	0.452189	0.5%	1.5%
p_2 : 0	0.552607	0.547811	0.9%	1.2%
1	0.447107	0.452189	-1.1%	1.5%
p_5 : 0	0.549523	0.547811	0.3%	1.1%
1	0.450277	0.452189	-0.4%	1.3%
p_7 : 0	0.649317	0.649456	-0.02%	0.6%
1	0.350330	0.350544	-0.06%	1.1%
p_{20} : 0	0.783583	0.783515	0.009%	0.3%
1	0.216066	0.216485	-0.2%	1.1%
p_{23} : 0	0.575953	0.576073	-0.02%	0.8%
1	0.423694	0.423927	-0.05%	1.1%

The simulation results (token distribution in the SPN in Figure 4) were compared with the results obtained from analytical solutions from the package. Table 2 shows the comparisons in terms of estimated values, relative errors and precisions of confidence intervals. The parallel simulation results were obtained from 18 independent runs for the estimated values and 95% confidence intervals.

It can be seen from the above that the algorithm results compare favorably with the analytical results. The errors are mainly due to the relatively shorter simulation runs, which is particularly indicated by the larger precision in estimating very small probabilities. For the further studies, several examples need to be simulated to assess the efficiency of the algorithm and overhead due to state savings, rollbacks and communication delays. Nevertheless, we believe the algorithm offers an important step towards automating the process of parallel simulation by letting the user specify only a partition of his model. We believe the partitioning problem can also be automated and optimized. These are important problems for future research in this area.

References

[1] D. R. Jefferson, "Virtual Time", *ACM Transactions on Programming Languages and Systems*, Vol.7, No.3, pp.404–425, July 1985.

[2] F. J. Kaudel, "A literature Survey on Distributed Discrete Event Simulation", *Simuletter* 18(2), pp.11–21, June 1987.

[3] D. B. Wagner, and E. D. Lazowska, "Parallel simulation of queueing networks: limitations and po-

tentials," *Proceedings of the 1989 SIGMETRICS conference*, Berkeley, CA, 1989.

[4] G. S. Thomas "Parallel Simulation of Petri Nets", M.S. Thesis in progress, Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195.

[5] K. Chandy, and J. Misra "Distributed simulation: a case in design and verification of distributed programs," *IEEE Trans. on Software Engineering*, 5(5):440-452, 1979.

[6] G. Chiola " A Graphical Petri Net Tool for Performace Evaluation," *Proceedings of the Third International Workshop on Modeling Techniques and Performance Evaluation*, AFCET, Paris, March 1987.

[7] H.H.Ammar and R.M.S.Islam, "Time Scale Decomposition of a Class of Generalized Stochastic Petri Net Models", *IEEE Transactions on software engineering* Vol.15, No.6, 1989.

[8] Randell, "System structure for software fault tolerance," *IEEE Trans. on Software Eng.*, vol. SE-1, no. 2, June 1975.

[9] K.G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. on Software Eng.*, vol. SE-10, no. 6, Nov. 1984.

[10] H.H.Ammar and Su Deng, "Distributed Discrete Event Simulation Using Time Scale Decomposition," Proceedings of the 1991 Workshop on Parallel and Distributed Simulation, Society of Computer Simulation, Anaheim, CA, January 1991.