Performance of a Distributed Simulation of Timed Colored Petri Nets with Fine-Grained Partitioning

Michael Knoke, Felix Kühling, Armin Zimmermann, Günter Hommel
Technische Universität Berlin
Real–Time Systems and Robotics
Einsteinufer 17, D–10587 Berlin, Germany
{knoke, felixyz, azi, hommel}@cs.tu-berlin.de

Keywords: Distributed simulation, Petri nets, performance

Abstract

Powerful grid and cluster computers allow efficient distributed simulation. Optimistic simulation techniques have been developed which allow for more parallelism in the local simulations than conservative methods. However, they may require costly rollbacks in simulation time due to dependencies between model parts that cause violations of global causality. Different notions of time have been proposed to detect and remedy these situations. Logical time (or Lamport time) is used in many present-day distributed simulation algorithms. However, high-level colored Petri nets may contain global activity priorities, vanishing states, and global state dependencies. Thus virtual time is not sufficient to maintain the global chronological order of events for the optimistic simulation of this model class. The paper presents a new approach that guarantees a correct ordering of global states in a distributed Petri net simulation. A priority-enhanced vector time algorithm is used to detect causal dependencies. This enables a fine-grained partitioning which is capable of performing precise rollbacks and helps to detect global priority conflicts. Some Petri net examples are used to present performance values. The influences of structural model characteristics of these examples are discussed in detail.

1 INTRODUCTION

Stochastic Petri nets (PN) have been widely used for modeling the behavior of systems where synchronization of processes is crucial [1]. They provide a graphical representation and are able to represent discrete events as well as (stochastic) timing. Our simulation framework uses a variant of *colored* Petri nets (CPN) [2].

Real world systems consist of parts widely showing autonomous behavior but cooperating or communicating occasionally. This inherent concurrency and required synchronization can be modeled adequately using PNs. Distributed Petri net simulation (DPNS) can exploit this inherent parallelism efficiently using grid- and cluster computers. Hence, a partitioning algorithm is required that decomposes the model such that heavily communicating elements are not separated. Each decomposed PN submodel is assigned to a *logical process* (LP) that performs the simulation on a physical processor. A logical clock that denotes how far the simulation has progressed is assigned to each LP as well. LPs communicate using timestamped messages [3].

There has been significant work in the area of distributed simulation of PNs in the past few years. Almost all proposed algorithms assume a virtual time with an arbitrary high resolution to eliminate isochronous events. Some model specific activities can also cause events with the same virtual time, even for an assumed infinite resolution of time. Some of the activities in high-level PNs are:

- immediate transitions resulting in state changes without virtual simulation time progress
- deterministic transitions that have a deterministic delay for state changes
- time guard functions which trigger state changes at a certain point in time

These properties of high-level PNs are either not allowed or limit the distribution to LPs, so that they must be sequentially processed. Nicol and Mao [4] have contributed one of the most complete publications on distributed simulation of PNs, showing this limitation in each presented algorithm. Today's most recent research results as reported by Furfaro et al [5] or Ya-Li Wu et al [6] doesn't deal with this problem but introduce new consolidated findings for Time Warp based DPNS. It is obvious that in these cases the event ordering is simple and most research is focused on preferably good partitioning algorithms, early rollback detection, and agent based technologies. PN models

for real world systems, such as detailed workflow modeling, may contain more than 50 percent timeless or deterministic activities.

A basic problem of distributed simulation is to avoid causality errors. Correctness of simulation can only be ensured if the (total) event ordering as produced by a sequential simulation is consistent with the (partial) event ordering due to distributed execution. Indeed, Jefferson [7] recognized this problem to be the inverse of Lamport's logical clock problem [8], i.e. providing clock values for events occurring in a distributed system such that all events appear ordered in logical time.

Lamport's algorithm allows to maintain time ordering among events [9]. However, a mapping from Lamport time to real time is not possible. Furthermore it is not sufficient to characterize causal relationships between events. But the detection of causal relationships between events is indispensable for transition priorities. Otherwise it is not possible to sort concurrent and independently fired events whose occurrence is based on a different priority. The Lamport time would impose an artificial order independent of their priority.

A logical time that characterizes causality and can be used to remedy last named problems is the *vector time* (VT) proposed by Mattern [10] and Fidge [11]. The causal relationships between events can be determined from their corresponding VT values. VT allows to detect indirect dependencies, that means comparing two VTs of different events provides information on whether these events are causally dependent and if so, which event depends on which one. This has the following advantages in the context of DPNS:

- concurrent events can be identified and sorted by their priorities
- a very fine-grained model partitioning allowing deterministic and zero-firing times for output transitions of LPs is possible
- precise recovery of local LP states based on external events
- no need to solve equal Lamport time stamps

Many different high-level colored PN model classes, our class as well, allow different priorities for immediate transitions. That means if two events could be generated at the same simulation time, the higher prioritized event is permitted first and may disable the second event through its occurrence. An example of a simple PN model is presented in Fig. 1. Transitions *Order* and *SendPart* are concurrently enabled and have different priorities, so that *Order* is processed first because of its higher priority. A sequential simulation is simple but a distributed simulation where both transitions fire optimistically, requires that the events are subsequently ordered by their priorities.

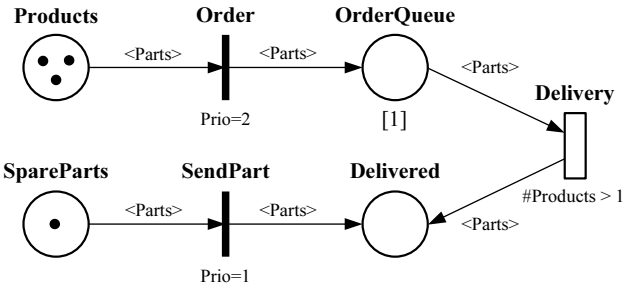

Figure 1. Example of a high-level colored Petri net model

The main motivation for our work is to realize a DPNS that is applicable for business process models. An arbitrarily low resolution of simulation time and a high number of prioritized isochronous state changes are typical characterizations of these models. Result measures are calculated by a subset of the global state which can not be identified by currently available DPNS algorithms. Petri nets, allowing transition priorities or dependencies on remote states can not be simulated by Time Warp based mechanisms. This paper presents a new logical time scheme for high-level PNs which has significant advantages for partitioning with less structural limitations than presented in [4]. Basic principles and a proof of correctness for a definite ordering of isochronous events have been proposed by Knoke et al. [12]. Our extensions to the logical time fulfill today's requirements for flexibility and maximum scalability for typical real world PN models. Some substantial performance measures for different model characteristics will be shown in this paper. These examples will not compare performance measures with any of the numerous Time Warp variations for distributed simulation of PNs. Optimistic simulation of high-level PNs is, in contrast to parallel discrete event simulation (PDES), heavily dependent on the abilities of the underlying net class. It is always possible to design PN models perfectly fitting a given distributed simulation algorithm. Our objective in this paper is to show and evaluate new algorithms for partitioning and distributed event processing based on a new logical time scheme that opens new possibilities for DPNS performance optimization.

The paper first presents our new partitioning approach in Sect. 2. The subsequent Sect. 3 introduces a logical time scheme for prioritized globally ordered states. A performance analysis together with a discussion about structural influences is presented in Sect. 4 and finally concluding remarks are given in Sect. 5.

2 A NEW FINE-GRAINED PARTITIONING APPROACH

The simulation is composed of N sequential event driven LPs that do not share memory and operate asynchronously in parallel. Unlike in other optimistic DPNS algorithms (e.g. introduced in [13]), an *atomic unit* (AU) is defined as the smallest indivisible part of the model, whereas a LP consists of one or more of these AUs. The basic architecture and formalism of the LPs and AUs used in this paper is:

- The smallest indivisible part of the model is an atomic unit AU.
- A transition T_i is inseparably linked with all of its input places ${}^\bullet T_i$ and constitutes an atomic unit AU. This can lead to situations where more than one transition will be assigned to one AU, namely if a place has several output transitions.
- Transitions without input places are constituting an own AU.
- At least one AU is assigned to every LP_i which is running as a process on one physical node N_i.
- AUs containing a single transition without input places are assigned to the LP that contains one of its output places. Otherwise it could generate events too far in the future and slow down the simulation.
- A communication interface attached to the LPs is responsible for the propagation of messages to the remote LPs and to dispatch incoming messages to local AUs. AUs on the same LP communicate directly to avoid additional message overhead.
- Each LP_i, AU_j has access to a partitioned subset of the state variables $S_{P,i} \subset S$ and $S_{U,j} \subset S_{P,i}$, disjoint to state variables assigned to other LPs, AUs. State variables of LP_i are the set of state variables of all local AUs $S_{P,i} = \bigcup S_{U,j} (\forall j)$.
- The simulation of local AUs is scheduled within each LP in a way that avoids local rollbacks.

The three basic items for event-driven DPNS are state variables which denote the state of the simulation model, an event list that contains pending events, and a simulation clock which keeps track of the simulation's progress. All of these parts have been integrated into the AUs. Only two basic messages are required for simulation progress of AUs: *positive event messages* for token transfers and *negative event messages* to perform a rollback to an earlier simulation time.

A fine-grained partitioning and a discrete storage of processed states have a number of advantages for DPNS. First of all, in contrast to existing DPNS algorithms, e.g. described by Chiola and Ferscha [14], a rollback of complete LPs will not happen. Each AU has it's own virtual simulation time and stores its state for each local event independently from other AUs. This characteristic is essential for migration to other LPs at runtime which is part of future work. AUs can restore their state accurately for a given simulation time and send rollback messages to other AUs if they are affected by this rollback. Thus, rollbacks are much more precise and unnecessary rollbacks are prevented if independent AUs are simulated by a single LP. Memory consumption is lower than the classical LP approach because rarely executing AUs don't need to save their states until their own next activity.

Very important for collecting the result measures is the storage of all processed states. This storage mechanism allows to revert exactly to a given logical time without needing to resimulate already simulated sequences. In case of a rollback the last valid state is found with absolut precision. The disadvantage of a higher memory consumption is compensated by the much smaller size of AUs.

The mapping of AUs to LPs is a key factor for the performance of the new DPNS, but is part of our current work and not shown here. There are much more possibilities than in currently available Time Warp partitioning algorithms. We assume a manually chosen optimal mapping for the example models in Sect. 4.

The amount of storage used for state-saving grows as the simulation progresses, also referred to as the "The Limited Memory Dilemma" [13]. Jefferson [7] observed that at any real time there exits a global virtual time, GVT, such that all saved local states earlier than GVT are confirmed and will never be annihilated by a rollback. Thus, result measures can be collected and written out. After this the storage used for saving information with timestamps earlier than GVT can be reclaimed. We are using a GVT algorithm based on Samadi's algorithm [15] to calculate the global virtual time. A central GVT manager (in our case the first assigned node of the cluster) requests the oldest unacknowledged message of each AU, calculates the GVT and broadcasts the computed value to all AUs such that they can purge their state queues.

For this fine-grained partitioning a correct implementation of causal dependencies is indispensable. A new logical time scheme that fulfills these requirements is presented in the next section.

3 A LOGICAL TIME SCHEME FOR PRIORITIZED HIGH-LEVEL DISTRIBUTED PN SIMULATION

In this section a new logical time scheme for DPNS is presented and studied in detail. As per description in Sect. 1 it is essential for a correct ordering of states

if model characteristics allows prioritized transitions and isochronous concurrent states. A distributed simulation is correct if its simulation results match the results of a traditional, single process simulator. Such sequential simulations process events in the order that takes the simulation time and the event priority into account. As a consequence we can conclude that a DPNS is correct if each AU processes events in the traditional sequential manner and if incoming events are sorted exactly as they would be generated by a single process simulator. The following section presents an extended logical time to fulfill these demands.

3.1 Event Priorities

For PN simulations on a single processor it is sufficient to have one *simulation clock time* with an arbitrarily low resolution. All activities are performed in succession and are responsible for the time increment. The simulated order of events is identical to the order in which they are simulated. Conflicts of concurrent activities are resolved by priorities or by random selection. Concerning distributed PN simulation, the simulation clock time can be used to order events if they are not isochronous. In other cases the VT is used. If VT detects two concurrent events they are sorted according to their firing priority.

In a high-level PN, immediate transitions have a priority greater than 0 and timed transitions have an implicit priority of 0. These priority values must be valid across AU borders, that means if transitions on different AUs are concurrently firing isochronously, the corresponding events must be ordered by their priority. Among identically enabled transitions one is chosen to fire first non-deterministically. For distributed simulation this approach is not applicable because of consistency reasons. Independent random generators on the AUs cannot guarantee the same ordering. Therefore we have decided to define a new *global event priority* (GEP) that includes the AU number into the priority value to determine an explicit relation for two equal event priorities. GEP forces the same global event ordering for concurrent events with different event priorities as a sequential simulation, but events with the same priority are ordered by the AU number in which they are created. It is calculated as follows:

$$GEP := P_E * N_{AU} + i_{AU} \quad \begin{array}{ll} P_E: & \text{event priority} \\ N_{AU}: & \text{AU count} \\ i_{AU}: & \text{current AU no.} \end{array} \tag{1}$$

Calculating the event priority P_E from the transition priority is nontrivial. The following order would be achieved by a sequential simulation of the model in Fig. 2: $T2 \rightarrow T4 \rightarrow T1 \rightarrow T3$. $T0$ is firing first and afterwards $T1$ and $T2$ are simultaneously enabled but $T2$ fires because of its higher priority. Now, without any simulation time elapsed, $T1$ and $T4$ are enabled and $T4$ fires. Subsequently $T1$ and $T3$ fire in succession without taking the priority values into account. This example looks simple but it is observable that in case of a distributed simulation the firing order requires global knowledge.

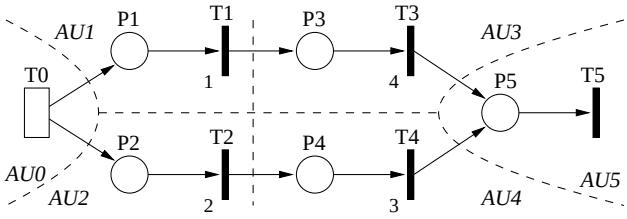

Figure 2. An example for transition priorities

An optimistic distributed simulation doesn't need to resolve this priority problem when it appears but at the time when affected tokens are inserted into a place. This happens if at least two concurrent isochronous tokens must be ordered according to their priority. In the example above it is $AU5$ which has to sort the concurrent events from $AU3$ and $AU4$. If the priority of the last fired transition was directly used to calculate the GEP it would give the token from path $T1 \rightarrow T3$ a higher order of precedence in the event queue because it was last fired from $T3$ which has a higher priority than $T4$.

To get the correct result for two concurrent events it is important to create a priority path (herein after called *critical path*) for each event. This path starts at the AU where both events must be sorted, $AU5$ in the example. All preceding immediate transitions which have triggered this event must be on this path, because their priority has an impact on the event order. It can be shown that the path contains all preceding transitions up to the last immediate transition without input places or the last timed transition, which is $AU0$ in the example. All priorities on each path must be considered for later event ordering. For two concurrent events the minimum priority P_{min} of each priority path is decisive because the transition with the lowest priority delays the propagation of an event until no other transition with a higher priority on other paths can fire. Using the minimum priority on both paths would deliver the correct result ($P_{min_{T1,T3}} = 1, P_{min_{T2,T4}} = 2$).

It is possible that both minimum priorities are equal if the paths share a common AU where no common predecessor event has taken place. In that case this AU has randomly defined the order. Figure 3 is one example. For n AUs it had to be AU_i with $i = p_{min} \ mod \ n$, as derivable through (1). The order is then explicitly observable by the corresponding VT

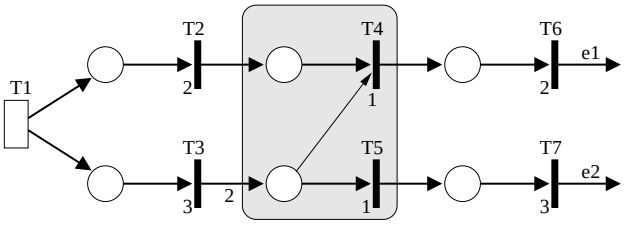

Figure 3. An example for equal minimal path priorities

component. In a sequential simulation transition $T1$ fires at first followed by transitions $T3$ and $T2$. $T4$ and $T5$ have the same priority, so that one of them is randomly selected. Afterwards either $T6$ or $T7$ fires. Regardless of which is selected, the order of events e_1 and e_2 has been determined by the gray marked AU.

Note that the priority of one event depends on the event it is compared to because the common predecessor event depends on both events. Therefore we also call it *relative event priority*.

Computationally, an AU-sized vector of the last firing priority of each AU is needed for calculating the minimum priority on the critical path. Events within an AU are always ordered sequentially, so it is not required to store the priorities of all transitions. This *priority vector* $p(e)$ has to be assigned to each event e. It is defined as follows:

$$p(e)_i = \begin{cases} \infty & \text{in case that } AU_i \text{ is not on the critical path} \\ \text{else} & \text{the minimum priority of all preceding events on the critical path of } e \text{ in } AU_i \end{cases} \tag{2}$$

To follow the path of AUs that a token has taken and to compute the minimum priority of this path, it is merely required to compute the minimum value of the priority vector. For this, only components of this vector should be used which belongs to events after the last common predecessor event, if this exists. It is a precondition that all components of this vector are set to the infinite value on initialization and if a timed transition fires. A mechanism to find the AUs on the critical path, on which a common event occurred, is required. These AUs can be obtained by comparing the VT components. Same VT components of both events ($VT(E_1)_i = VT(E_2)_i$) indicate a common event on AU_i. Components not corresponding to AUs on the critical path have no effect for computing the minimum priority because their priorities are infinite.

Assuming that two concurrently fired isochronous events arrived at $AU5$ in Fig. 2 with $(VT), [P_E]$:

$$E_1 := (1,1,0,1,0,0), [\infty,1,\infty,4,\infty,\infty]$$
$$E_2 := (1,0,1,0,1,0), [\infty,\infty,2,\infty,3,\infty]$$

The VT indicates that both tokens are concurrent, but the minimum priority of E_1 is lower than the priority of E_2. As a result E_1 must be sorted after E_2.

3.2 Compound Simulation Time for Globally Ordered States

A limited resolution of the simulation clock time and the occurrence of prioritized isochronous events make demand for a sufficient logical time scheme. It must be composed of the simulation clock time, a logical time, namely the VT, and the GEP introduced in Sect. 3.1. This compound *simulation time* (ST) (3) can be used then to guarantee a correct ordering of global states. An appropriate ordering relation has been defined (4). We have proven the transitivity of the relation, which is not obvious due to the relative GEP values.

$$\begin{aligned} ST =& (T, V, G) \\ & T: \text{ Simulation clock time} \\ & V: \text{ Vector time} \\ & G: \text{ Global event priority} \end{aligned} \tag{3}$$

$$\begin{aligned} u \leq v \quad \Leftrightarrow \quad & (T_u < T_v) \vee \\ & ((T_u = T_v) \wedge ((V_u < V_v) \vee \\ & (V_u \parallel V_v \wedge G_u \geq G_v))) \end{aligned} \tag{4}$$

Figure 4. Compound simulation time

A common problem to all distributed simulations is to determine a global state out of the numerous local states of each LP. In the context of DPNS these global states are required twice. *Global guards* and place capacities pose a condition related to the global state as visible by transitions *Delivery* and *Order* in Fig. 1. Transition *Delivery* has a global guard counting the number of tokens in place *Products*. Furthermore the firing of transition *Order* is limited by the number of tokens in place *OrderQueue*. Obviously, in an optimistic distributed simulation these conditions can not be verified immediately. This has to be done when the corresponding events are merged into the event list of the depending place. The second reason for the importance of global states are global result measures which may depend on several places of independently processing AUs or LPs. In [10] the concept of causally *consistent cuts* for time diagrams is proposed. A global state is composed of the local states of all processes at the time of the "cut events". This

implies that global predicates don't change over time. But the order of concurrent events is undetermined by causality, so that this concept can not be applied for DPNS.

3.3 Optimized Cancelation mechanism

Chetlur and Wilsey (2001) have proposed a causality representation and cancelation mechanism for Time Warp simulations [16]. Conventionally, rollbacks are informed through anti-messages with the timestamps specifying the rollback time of the LPs. Rollbacks can occur frequently and may be cascaded and inter-related. In contemporary Time Warp simulators, time representations generally maintain only the local simulation time and do not usually carry information about causal relations between rollbacks and the associated events. The cause for such cascading and inter-related rollbacks is due to the fact that the events that are causally dependant on the events rolled-back are not identified at the time of a causality error. However, our logical time representation is designed to carry causal information that can be exploited during rollback to accelerate the cancelation process. It can save a huge amount of computation and communication time by ignoring events that will be rolled-back eventually. Furthermore it guarantees a lifelock free simulation progress, provided that in case of a rollback the simulation is rolling back to the precise point in time.

Chetlur and Wilsey introduced *Total Clocks* containing a virtual time component (as a global one dimensional temporal coordinate system) and a vector counter component (similar to VT). Using these clocks as additional logical time mechanism in Time Warp simulations permits an accurate detection of cascaded and inter-related rollbacks. But it can be seen as a big drawback that the additional costs of this logical time are not used to enhance rollback accuracy nor correct distributed simulation of priorities or handling of distributed measures.

The new logical time scheme presented in this paper already provides a causality representation for an optimized cancelation mechanism. Cascading events are prevented by removing events which are causally related to the events which are rolled-back due to a rollback (also called cancelation) message. This is due to the fact that, assuming aggressive cancelation strategy, the events causally dependant on the rolled-back event will eventually be rolled-back. In addition, early recovery operations such as restoring state and ignoring events that will be rolled-back can be performed for rollbacks that are inter-related. A signature is attached to each cancelation message containing the AU number and the current VT of the AU which has triggered the rollback. This helps to identify related cancelation messages and hence inter-related and cascading rollbacks if the same signature is used during rollback propagation. An example that shows how many rollbacks are prevented by this algorithm is presented as "canceled messages" in Fig. 6.

4 TESTS

In the course of our study we have designed numerous models to verify our implementation of the new logical time scheme and to compare performance with a sequential simulation. The AU approach allows a truly distributed simulation of simple models to exploit parallelism. All experiments have been conducted on a 16 node, dual Intel Xeon 1.7 GHz processor, Linux cluster with 1 GB memory on each node. SCI has been used as a high-speed network connected as a 2D-Torus. Only one processor on each cluster node has been used for all experiments to simplify comparison of the results. To discuss the performance of the new mechanism we abandon a demonstration of best performing models. It is obvious that complex models with numerous places and transitions and with less synchronization between model parts will achieve the highest speedup in a distributed environment. But because of their complexity these models are not qualified to discuss here. Otherwise, a distributed simulation of simple models on fast processors will never achieve good overall performance. As a consequence we discuss very small models but we simulate complexity through an additional delay which is presented later. This helps to demonstrate performance potentials and limitations.

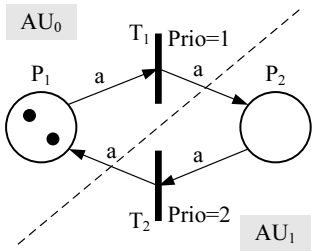

Figure 5. Example petri net for testing rollback performance

Figure 5 shows a very simple Petri net model which is absolutely not qualified for a distributed simulation. However it provides useful information about the worst case performance. If this model is divided into two parts (LPs), each part is likewise input and output of the other part and thus, each firing event must be sent to the other LP. In consequence of the network latency each LP is processing all local events prior to receive a response regarding the first message sent. The higher priority of transition T_2 causes a rollback of AU_0 in most cases. Numerous cascading rollback events can occur, because if AU_0 detects the priority conflict after receiving the first token from AU_1, it sends a negative message to AU_1 but this AU has already fired the sec-

ond token in the majority of cases. This could end up in infinite rollbacks, but will be immediately detected by the algorithm described in Sect. 3.3. All of this leads to a poor absolute performance as seen in Fig. 6. The first line of both tables contains the transition firing count. Rollback attempts are counted in line two, whereas some of these rollbacks have been canceled (as observable in line three) because of cascading or inter-related rollback events. Line four contains the maximum size of the AU state history over time. This history is cleared frequently up to the GVT. Both, the simulation time of a sequential run or rather the speedup is displayed in the last line of this table. The model has been simulated using three different configurations: containing two tokens as depicted in Fig. 5, containing 16 tokens, and containing two tokens but with an additional firing computation delay. It is obvious that this delay ($1,000,000$ iterations of an empty loop) feigns more activity inside one LP before the token is moved to the next LP. It may be seen as if each token has to be fired by some more transitions (nearly 3 transitions for the currently used delay) inside the LP or alternatively the output inscriptions require more computational time for processing complex token attributes. Hence, the speedup using this delay is much more meaningful if simple models are being simulated as done here. Otherwise the message overhead is the limiting factor. Without delay it is observable that a distributed simulation on two cluster nodes needs about twice the time due to the above named characteristics. The delay eliminates rollbacks nearly completely for this model and leads to a speedup of 1.7 compared to the sequential simulation. The target LP is always capable of processing the first token, so that the source LP detects the priority conflict before sending the second token.

The next performance analysis is illustrated by the distributed simulation of a Kanban system consisting of five machines in line [17]. Figure 7 shows the respective Petri net model along with the AU partitioning. It has three dependent loops. The rollback frequency varies depending on the used AU to LP association. For these experiments the best association was chosen manually.

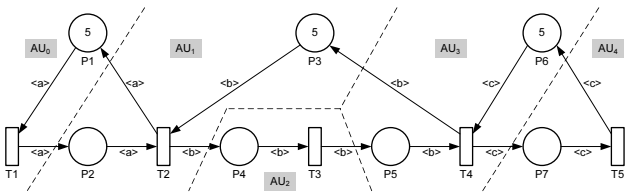

Figure 7. Kanban Petri net model

Some performance values for a distributed simulation of this model running on a different number of cluster nodes are diagramed in Fig. 8. This diagram

1 Cluster node	2 tokens no delay	16 tokens no delay	2 tokens delay
Firing count	120961	120961	120961
Rollback count	0	0	0
Canceled messages	0	0	0
Stored AU states	2665	2665	1462
Simulation time (s)	99	73	167

2 Cluster nodes	2 tokens no delay	16 tokens no delay	2 tokens delay
Firing count	241896	241896	122087
Rollback count	120958	120958	1232
Canceled messages	60454	60454	371
Stored AU states	2748	2748	4853
Speedup	0.49	0.45	1.70

Figure 6. Performance measures

shows the speedup of a distributed simulation on several LPs compared to a sequential simulation. A small speed improvement can be observed on two or three LPs. These cases benefit from a suitable partitioning with some sporadic rollbacks only, but the speedup is limited by the fast processor speed. On four LPs the partitioning requires costly rollbacks. $75,000$ rollbacks occurred during $180,000$ transition firing events. An additional firing computation delay as mentioned above feigns a more complex model and is equivalent to a Kanban system consisting of about 15 machines. The speedup then improves considerably up to 2.

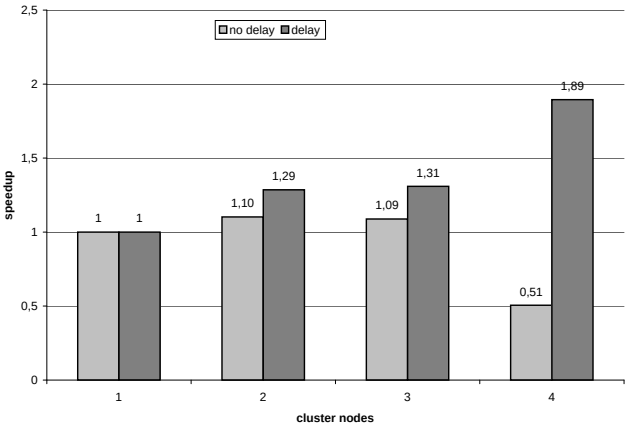

Figure 8. Performance measures of the Kanban model

The next model in Fig. 9 is a modified Kanban model and shows the impact of depending loops for a distributed simulation of small models. Compared to the model in Fig. 7 each loop is replaced by a new transition that generates new tokens. The firing delay

of these transitions defines the possible speed of the associated machines (transitions). There is no longer a backward synchronization between machines.

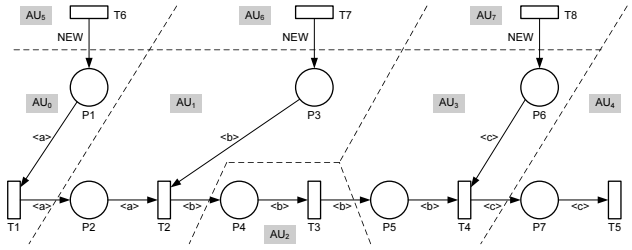

Figure 9. Kanban Petri net model without loops

A distributed simulation of this model gains speed through the decoupled model parts. Tokens have to be processed further on in succession. That means, the second LP can simulate a token not until the first LP has sent this token. In spite of a furthermore unfair arrangement between computing time and network load, the model performs better as visible in Fig. 10. A simulation on four cluster nodes at full speed is about two times faster than on a single node. 484,062 transitions have fired in 2,425 seconds while 2,862 rollbacks took place. The additional delay (equivalent to a Kanban models consisting of 15 machines) results in a much better speedup of 3 on four LPs.

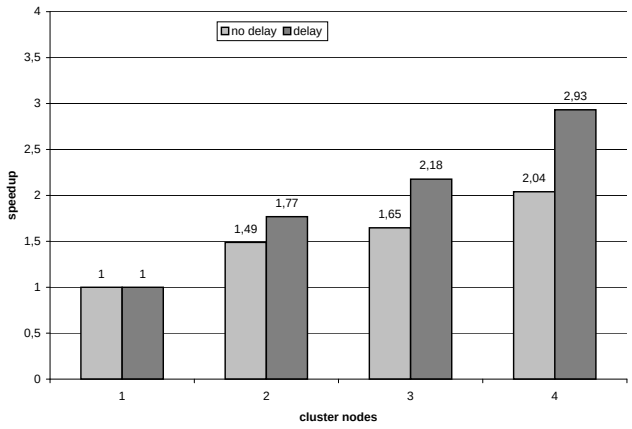

Figure 10. Performance measures of the modified Kanban model

Three examples introduced in this chapter have shown partially significant speedup for a distributed simulation of high-level Petri net models. The new logical time mechanism qualifies for a fine-grained partitioning and can be applied even to simple models. We verified that the result measures are correct and the order of processing is exactly the same as performed by a sequential simulation. More complicated models, which have been developed for different research projects for global operative business companies, are already successfully tested with good speedup results but not shown here because of their complexity.

5 CONCLUSIONS AND FUTURE WORK

This paper presented a new mechanism for distributed simulation of high-level Petri Nets. The Petri net model is decomposed into atomic units which have their own virtual time and state history. Many potential advantages for the distributed simulation arise from this approach: a better partitioning flexibility and efficient rollbacks.

In addition we introduced the notion of prioritized logical time which allows for a mapping between simulation clock and logical time. Applied to high-level PNs, this logical time is sufficient to allow a fine-grained partitioning not possible with Lamport's logical time. We succeeded in defining a total ordering relation which allows global ordering of events and the identification of global states of the distributed simulation.

The performance of this new mechanism has been shown on different Petri net models. In the majority of cases the speedup was very good considering the low model complexity. Structural model characteristics which have a bearing on distributed performance have been discussed in detail.

Compared to the well known Time Warp mechanism it is obvious that the operational overhead is slightly higher. This is compensated by an optimized cancelation mechanism, more precise rollbacks and a lower state history overhead. The biggest advantages over the Time Warp mechanism are the ability to correctly simulate prioritized immediate transitions without limiting partitioning flexibility and the possibility to evaluate conditions based on the global state of the model, such as global guards, place capacities and result measures.

Future work will focus on runtime load balancing with respect to the capabilities of a fine-grained partitioning. The free association of AUs to LPs allows a more flexible partitioning than currently available for high-level PNs. A dynamic migration of AUs to other LPs will require low operational expense because of the autonomous AU structure.

REFERENCES

[1] Zimmermann, A.; Freiheit, J.; Huck, A. 2001, "A Petri net based design engine for manufacturing systems." International Journal of Production Research, 39, no. 2: 225–253.

[2] Jensen, K. 1997, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1 : Basic Concepts (2nd Edition)*. EATCS Monographs on Theoretical Computer Science,

Springer-Verlag.

[3] Fujimoto, R. 1993, "Parallel and distributed discrete event simulation: algorithms and applications." In *WSC '93: Proceedings of the 25th conference on Winter simulation*, ACM Press, Los Angeles, CA, USA, 106–114.

[4] Nicol, D.; Mao, W. 1995, "Automated parallelization of timed Petri-net simulations." Journal of Parallel and Distributed Computing, 29, no. 1: 60–74.

[5] Furfaro, A.; Nigro, L.; Pupo, F. 2002, "Distributed simulation of timed coloured Petri nets." In *Proceedings. Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT'02)*, IEEE Computer Society, 159–166.

[6] Wu, Y.; Zeng, J.; Sun, G. 2002, "Distributed simulation algorithms of generalized differential Petri nets." In *Proceedings of the 2002 International Conference on Machine Learning and Cybernetics (ICMLC02)*, IEEE Computer Society, Beijing, China, 1013–1017.

[7] Jefferson, D. 1985, "Virtual time." ACM Transactions on Programming Languages and Systems (TOPLAS), 7, no. 3: 405–425.

[8] Lamport, L. 1978, "Time, clocks, and the ordering of events in a distributed system." Communications of the ACM, 21, no. 7: 558–565.

[9] Zeng, Y.; Cai, W.; Turner, S. 2003, "Causal Order Based Time Warp: A Tradeoff of Optimism." In *Proceedings of the 2003 Winter Simulation Conference*, New Orleans, LA, USA.

[10] Mattern, F. 1989, "Virtual Time and Global States of Distributed Systems." In M. et. al., ed., *Parallel and Distributed Algorithms: Proceedings of the Workshop on Parallel and Distributed Algorithm*, Elsevier Science Publishers B.V.(North-Holland), 215–226.

[11] Fidge, C. 1991, "Logical Time in Distributed Computing Systems." Computer, 24, no. 8: 28–33.

[12] Knoke, M.; Kuehling, F.; Zimmermann, A.; Hommel, G. 2004, "Towards Correct Distributed Simulation of High-Level Petri Nets with Fine-Grained Partitioning." In J. C. et. al., ed., *2nd Int. Symposium on Parallel and Distributed Processing and Applications (ISPA'04)*, IEEE, Springer LNCS 3358, Hongkong, China, 64–74.

[13] Ferscha, A.; Tripathi, S. K. 1994, "Parallel and Distributed Simulation of Discrete Event Systems." Tech. rep., University of Maryland, College Park, MD, USA.

[14] Chiola, G.; Ferscha, A. 1993, "Distributed Simulation of Petri Nets." IEEE Parallel & Distributed Technology: Systems & Technology, 1, no. 3: 33–50.

[15] Samadi, B. 1985, "Distributed Simulation, Algorithms and Performance Analysis." Tech. rep., University of California.

[16] Chetlur, M.; Wilsey, P. 2001, "Causality representation and cancellation mechanism in time warp simulations." In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, IEEE Computer Society, Lake Arrowhead, CA, USA, 165–172.

[17] Singh, N. 1995, *Systems Approach to Computer-Integrated Design and Manufacturing*. John Wiley & Sons Inc., pages 630–631.