A Logical Process Simulation Engine with Probabilistic Optimism*

A. Ferscha† and G. Chiola*

† Institut für Angewandte Informatik und Informationssysteme, Universität Wien
Lenaugasse 2/8, A-1080 Vienna, Austria

* Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, I-10149 Torino, Italy

Abstract

A *probabilistic* distributed discrete event simulation strategy is developed as a performance efficient compromise between the two classical approaches in parallel and distributed simulation, the *conservative* and the *optimistic* approach. It weakens the conservative "block until safe-to-process"-rule in a sense that if the time instant of the occurrence of an external event is in the time interval $[s, t]$, it allows progressing simulation up until $\widehat{t}(O)$, $s \leq \widehat{t}(O) \leq t$, where $\widehat{t}(O)$ is an estimate based on the arrival instants $O = (o_1, o_2, \ldots o_n)$ observed during a time window by some logical (simulation) process. Compared to the optimistic strategy it prevents from propagating incorrect computations too far ahead into the simulated future, and thus avoids unnecessary communication overhead by breaking rollback cascades as early as possible. Moreover, the arrival patterns observed in O are used to *adapt* the logical process to a synchronization behavior that is the best tradeoff among blocking and optimistically progressing with respect to the parallelism inherent in the simulation model.

The probabilistic protocol is described at a ready to implement level of detail, and statistical methods for the characterization of the arrival process and the prediction of next external event occurrence times are developed. The potential performance improvement of the new protocol upon prototype implementations of the two classical protocols that have been conducted for the iPSC/860 is discussed. The simulation models studied are timed transition Petri nets, for which sophisticated analysis and (sequential) simulation tools (GreatSPN) are at hand.

1 Introduction

Asynchronous, distributed simulation of time dynamical systems has historically progressed along two different front-lines: the *conservative* [Misr 86] and the *optimistic* [Jeff 85a] approach. A variety of amendments to these schemes is at hand today [Fuji 90], most of them developed to improve the performance of implementations by exploiting domain intrinsics of the simulated (*physical*) system. Common to both approaches is a partitioning of the simulation model into a set of submodels being simulated by communicating *logical processes* (LPs). Distributed discrete event simulation (DDES) employs message based communication for two reasons: first, to maintain causality relations among events across all participating LPs, and second, to synchronize the progress of work in the LPs with respect to those event causalities. Every LP performs the execution of a sequence of events, thus advancing a *local clock*. Two kinds of events have to be distinguished: *internal events* have causal impact only on the local *state* of some LP, and *external events* also affecting the local states of other LPs.

*This work was conducted while Alois Ferscha was visiting the Dipartimento di Informatica, Università di Torino during August 1993, supported by a grant of the Academic Senate of the University of Vienna.

The conservative scheme does not allow an internal event with timestamp t to be processed if there is the chance of an external event with timestamp $s < t$. The optimistic scheme on the other hand allows LPs to compute under the assumption of absence of external events; in the case that such an external event actually occurs, the LP invalidates the preliminary processing of internal events and starts over again taking the external event into account. The mechanism of invalidating processed events is known as *rollback*.

The propagation of external events in DDES is implemented in terms of time stamped messages exchanged among LPs. An implementation of the conservative scheme hence would force an LP to block the processing of internal events at time t, if it cannot be sure that there are pending messages with timestamp $s < t$. Implementations of the optimistic scheme would process internal events without respecting any bound on local time, and rolling back upon the receipt of a message timestamped in the local past. Implementations of the two DDES protocols have been undertaken for the simulation of large scale timed transition Petri nets (TTPNs). We have reported on our observations of the performance of simulating TTPNs on a variety of parallel architectures in [Chio 93b] [Chio 93c].

This work considers the two schemes as extremes in a spectrum of possibilities with respect to the treatment of external events. Consider that some external event ee_i occurs in some time interval $[s, t]$, and let the probability of the actual occurrence time $occ(ee_i) = t*$ be equally likely for all $s \leq t* \leq t$. Then the conservative simulation protocol could have produced real simulation work by progressing over s with probability $(1 - \alpha)$ during the time interval $[s, s + \alpha(t - s)]$. So at least for small values of α the conservative protocol seems overly pessimistic. The optimistic protocol on the other hand will proceed simulating internal events even after surpassing t. This is overly optimistic in the sense that every internal event with $occ(ie_i) > t$ will definitely have to be invalidated.

We therefore propose a DDES simulation protocol that is *probabilistic* in the sense that it weakens the strict safe-to-process rule of the conservative protocol (or restricts the optimistic protocol to eventually slow down progressing into the future) based on information that is extracted from the simulation model. We show how the probabilistic protocol compares to the traditional approaches in the case of DDES of TTPNs: when compared to the conservative strategy we observe a reduction of the LP blocking time, while when compared to the optimistic protocol we observe an early breaking of rollback cascades, thus reducing the communication overhead considerably.

The paper is organized as follows. In Section 2 we briefly recall our concept of distributed simulation of TTPNs. In Section 3 we work out motivations for the probabilistic protocol in terms of an easy to follow "toy" example. Section 4 describes the protocol. Section 5 is dedicated to several possibilities of adapting the LPs' behavior to the arrival process of external events based on history information collected during the execution. Conclusions are drawn in Section 6.

2 Discrete Event Simulation of TTPNs

Discrete event simulation (when executed sequentially) repeatedly processes the occurrence of events in simulated (virtual) time (VT) by maintaining a time ordered *event list* (EVL) holding time stamped events scheduled to occur in the future, a (global) *clock* indicating the current time, and *state variables* defining the current state of the system. A *simulation engine* (SE) drives the simulation by continuously taking the first event out of EVL (i.e., the one with the lowest time stamp), simulating the effect of the event by changing the state variables and/or scheduling new events in the EVL — possibly also removing obsolete events. This sequence is performed until some pre-defined *endtime* is reached, or there are no further events in the EVL.

A natural correspondence among events and transition firings is exploited in discrete event simulation (DES) of TTPNs. The occurrence of an event in the simulation system relates to the firing of a transition in the TTPN model. The EVL hence carries event entries $e_i = \langle t_k@FT \rangle$, announcing the firing of transition t_k at time FT, given that the firing does not become obsolete in the meantime (due to the occurrence of a conflicting transition). The state of the system is represented by the current marking M of the TTPN, which is being changed by the processing of an event, i.e. the firing of a transition. A state change is simulated by withdrawing the transition with the smallest time stamp from the EVL, by removing tokens from its input places (modification of M) and by depositing an appropriate number of tokens in its output places (again a modification of M). VT is incremented by setting it to the FT of the transition just simulated. The new marking, however, can enable new transitions or disable previously enabled transitions, such that the EVL has to be corrected accordingly: new enabled transitions are scheduled with their firing time to occur in the future by inserting them into the EVL, while transitions that became disabled are removed.

In order to simulate a TTPN model by a set of distributed, communicating logical processes we define a *logical process* to be a tuple $LP_k = (R_k, I_k, SE)$ where R_k is a topological *"region"* of some TTPN (i.e. a subset of the places, transitions and arcs), I_k is the LP's communication interface implementing the required message exchange among LPs, and SE is the simulation engine simulating R_k (either according to the optimistic or the conservative protocol). The set of all LPs $LP = \bigcup_k LP_k$ together with directed, reliable, FIFO communication channels $CH = \bigcup_{k,i\ (k \neq i)} ch_{k,i} = (LP_k, LP_i)$ constitute the *Graph of Logical Processes* $GLP = (LP, CH)$. Distributed versions of DES for TTPNs have been described in [Chio 93b] [Chio 93c] using partitioning criteria based on a preanalysis of the model structure. We avoid the presentation of the details here; instead in the next section we show the result of the partitioning in a simple case.

3 A Simple Model with Inherent Parallelism

Consider the TTPN depicted in the Figure 3 comprising two places and two transitions with stochastic timing and balanced firing delays ($\tau(\mathrm{T1}) \sim exp(1.00)$, $\tau(\mathrm{T2}) \sim exp(1.00)$). Assume that T1 and T2 apply infinite server semantics [Chio 93a]. Related to those firing delays a certain amount of model parallelism can be exploited when partitioning the net into two LPs to be assigned to individual physical processors in order to simulate the behavior of the TTPN in parallel. The individual TTNP regions of LP_1 and LP_2 are: $R_1 = (\{\mathrm{T1}\}, \{\mathrm{P1}\}, \{(\mathrm{P1}, \mathrm{T1})\}, \tau(\mathrm{T1}) \sim exp(\lambda_1 = 1.00))$, and $R_2 = (\{\mathrm{T2}\}, \{\mathrm{P2}\}, \{(\mathrm{P2}, \mathrm{T2})\}, \tau(\mathrm{T2}) \sim exp(\lambda_1 = 1.00))$,

Let the *future list* [Nico 88], a sequence of exponentially distributed random firing times (random variates), for T1 and T2 be as in table 2 (the instances were collected using the GreatSPN simulation engine [Chio 91]). The sequential simulation would then sequence the variates according to their resulting scheduling in virtual time units when simulating the timed behavior of the net. This sequencing stems from the policy of always simulating the internal event (transition firing scheduled in EVL) with smallest timestamp first, since otherwise the future of the simulation could have an impact onto the past (*causality errors*).

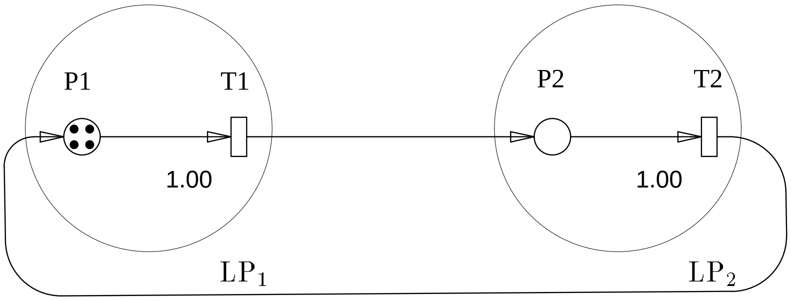

Figure 1: A small TTPN

In a DDES scheme this sequencing is related to the protocol applied to maintain causality among the events. Fur further arguing upon this small model now observe that the firing of a scheduled transition (internal event) always generates an external event, namely a message carrying a token with timestamp equal to the *local virtual time* LVT of the sending LP. On the other hand, the receipt of such a *token message* (external event) always causes a new internal event to the receiving LP, namely

Sequence T1	Sequence T2	Step	P1	Firing T1	P1'	P2	Firing T2	P2'
1.40297	0.12469	1	4	0.70543	3	0		
1.34856	0.09803	2				1	0.83012	0
1.07530	2.33729	3	4	1.07530	3			
0.70543	0.97464	4				1	1.17333	0
0.50610	0.33161	5	4	1.33622	3			
0.62534	0.25315	6	3	1.34856	2			
0.76501	0.63191	8	2	1.40297	1			
2.49070	1.84579	9				3	1.73458	2
0.69863	5.92625	10	2	1.79867	1			
etc.	etc.			etc.			etc.	

Figure 2: Sequence of Random Variates and Sequential DES

the scheduling of a new transition firing in the local EVL. Assuming now that latency of an external event is neglectible, i.e. a token produced by simulating the occurrence of firing T1 is available at P1 at (almost) the same time, then we see from the trace in Figure 2, that the first occurrence of T2 and the second occurrence of T1 could be simulated in a time overlapped way. The same is true for the second occurrence of T2 and the third occurrence of T1, etc. Hence, the model that we consider in Figure 3 provides inherent model parallelism. Exploitation of this model parallelism is the hope when applying one of the DDES protocols.

In a distributed simulation of the example net the two regions R_1 and R_2, which represent the submodels of the TTPN simulation model, are assigned to the two logical processes LP_1 and LP_2, such that $GLP = (\{LP_1, LP_2\}, \{ch_{1,2}, ch_{2,1}\})$, where the channels $ch_{1,2}$ and $ch_{2,1}$ are supposed to substitute the TTPN arcs (T1, P2) and (T2, P1) respectively. Both $ch_{1,2}$ and $ch_{2,1}$ carry messages containing tokens that were generated by the firing of a transition in a remote LP. So would $ch_{1,2}$ for example bring a message of the form $m = \langle 1, \mathrm{P2}, TT, \rangle$ from LP_1 to LP_2 on the occurrence of a firing of T1, in order to deposit 1 (first component of m) token into place P2 (second component of m) at time TT (third component). TT is called the *token time*, which is produced as a copy of the LVT of LP_1 at the instant of that firing of T1, that produced the token. We assume that I_k, the communication interface of LP_k takes care of converting tokens into messages and vice versa.

3.1 Observations under Conservative Simulation

A SE following the conservative approach (in further denoted by SE^{cons}) allows only the processing of *safe* (internal) events, i.e. the firing of transitions up to a LVT for which the LP has been guaranteed not to receive token messages (external events) with timestamp "in the past." The causality of events across LPs is preserved by sending timestamped token messages of type $m = \langle \#, P, TT \rangle$ (with $\# > 0$) in nondecreasing time stamp order, or by sending a promise $m = \langle 0, D, TT \rangle$ (*null message*) not to send a new message time-stamped earlier than TT. The events caused by token messages are in turn processed in nondecreasing time stamp order, such that all their effects will again produce messages with times in the future. The I_k on the input side maintains one input buffer IB[i] and a channel clock CC[i] for every channel $ch_{i,k} \in CH$ pointing to LP_k. CC[i] records the timestamp of the token message or null message. LVTW = $\min_i \mathrm{CC}[i]$ is the time window up until which LVT is allowed to progress by simulating internal or external events. Given that within that window neither internal nor external events are available to process, then LP_k *blocks* processing, and idles to receive new messages widening the time window. On the output side I_k collects token messages produced by SE^{cons} in individual output buffer OB[i] for every outgoing channel $ch_{k,l} \in CH$ to subsequent LPs LP_l.

The blocking of SE^{cons} as soon as LVTW is less than the occurrence time of the first event in EVL obviously has a severe impact on the parallel execution performance. This situation can be improved by the extent of *lookahead*, i.e. the lower bound on guaranteed LVT increment within some LP, that

LP$_1$ on Processor 1											
S	IB	P1	LVTW$_1$	B	LVT$_1$	Eventlist $\langle T1@\ \rangle$				OB	FC
0		4	0.00000		0.00000	0.70543	1.07530	1.34856	1.40297		
1		4	0.00000	*	0.00000	0.70543	1.07530	1.34856	1.40297	$\langle 0, P2, 0.70543\rangle$	
2	$\langle 0, P1, 0.09803\rangle$	4	0.09803	*	0.00000	0.70543	1.07530	1.34856	1.40297	$\langle 0, P2, 0.70543\rangle$	
3	$\langle 0, P1, 0.80346\rangle$	3	0.80346		0.70543	1.07530	1.34856	1.40297		$\langle 1, P2, 0.70543\rangle$	1
4	$\langle 0, P1, 0.80346\rangle$	3	0.80346	*	0.70543	1.07530	1.34856	1.40297		$\langle 0, P2, 1.07530\rangle$	
5	$\langle 0, P1, 0.80346\rangle$	3	0.80346	*	0.70543	1.07530	1.34856	1.40297		$\langle 0, P2, 1.07530\rangle$	
6	$\langle 1, P1, 0.83012\rangle$	4	0.83012		0.83012	1.07530	1.33622	1.34856	1.40297	$\langle 0, P2, 1.07530\rangle$	
7	$\langle 0, P1, 1.17333\rangle$	3	1.17333		1.07530	1.33622	1.34856	1.40297		$\langle 1, P2, 1.07530\rangle$	2
8	$\langle 0, P1, 1.17333\rangle$	3	1.17333	*	1.07530	1.33622	1.34856	1.40297		$\langle 0, P2, 1.33622\rangle$	
9	$\langle 0, P1, 1.17333\rangle$	3	1.17333	*	1.07530	1.33622	1.34856	1.40297		$\langle 0, P2, 1.33622\rangle$	
10	$\langle 1, P1, 1.17333\rangle$	4	1.17333		1.17333	1.33622	1.34856	1.40297	1.79867	$\langle 0, P2, 1.33622\rangle$	
11	$\langle 0, P1, 1.58937\rangle$	1	1.58937		1.40297	1.79867				$\langle 1, P2, 1.33622\rangle$	3
										$\langle 1, P2, 1.34856\rangle$	4
										$\langle 1, P2, 1.40297\rangle$	5
12	$\langle 0, P1, 1.58937\rangle$	1	1.58937	*	1.40297	1.79867				$\langle 0, P2, 1.79867\rangle$	
13	$\langle 0, P1, 1.65612\rangle$	1	1.65612	*	1.40297	1.79867				$\langle 0, P2, 1.79867\rangle$	
14	$\langle 1, P1, 1.73458\rangle$	2	1.73458		1.73458	1.79867	2.49959			$\langle 0, P2, 1.79867\rangle$	
15	$\langle 0, P1, 2.05182\rangle$	1	2.05182		1.79867	2.49959				$\langle 1, P2, 1.79867\rangle$	6
16	$\langle 0, P1, 2.05182\rangle$	1	2.05182	*	1.79867	2.49959				$\langle 0, P2, 2.22793\rangle$	
LP$_2$ on Processor 2											
S	IB	P2	LVTW$_2$	B	LVT$_2$	Eventlist $\langle T2@\ \rangle$				OB	FC
0		0	0.00000		0.00000						
1		0	0.00000	*	0.00000					$\langle 0, P1, 0.09803\rangle$	
2	$\langle 0, P2, 0.70543\rangle$	0	0.70543	*	0.00000					$\langle 0, P1, 0.80346\rangle$	
3	$\langle 0, P2, 0.70543\rangle$	0	0.70543	*	0.00000					$\langle 0, P1, 0.80346\rangle$	
4	$\langle 1, P2, 0.70543\rangle$	1	0.70543		0.70543	0.83012				$\langle 0, P1, 0.80346\rangle$	
5	$\langle 0, P2, 1.07530\rangle$	0	1.07530		0.83012					$\langle 1, P1, 0.83012\rangle$	1
6	$\langle 0, P2, 1.07530\rangle$	0	1.07530	*	0.83012					$\langle 0, P1, 1.17333\rangle$	
7	$\langle 0, P2, 1.07530\rangle$	0	1.07530	*	0.83012					$\langle 0, P1, 1.17333\rangle$	
8	$\langle 1, P2, 1.07530\rangle$	1	1.07530		1.07530	1.17333				$\langle 0, P1, 1.17333\rangle$	
9	$\langle 0, P2, 1.33622\rangle$	0	1.33622		1.17333					$\langle 1, P1, 1.17333\rangle$	2
10	$\langle 0, P2, 1.33622\rangle$	0	1.33622	*	1.17333					$\langle 0, P1, 1.58937\rangle$	
11	$\langle 0, P2, 1.33622\rangle$	0	1.33622	*	1.17333					$\langle 0, P1, 1.58937\rangle$	
12	$\langle 1, P2, 1.33622\rangle$	1	1.33622		1.33622	3.67351					
	$\langle 1, P2, 1.34856\rangle$	2	1.34856		1.34856	2.32320	3.67351				
	$\langle 1, P2, 1.40297\rangle$	3	1.40297		1.40297	1.73458	2.32320	3.67351		$\langle 0, P1, 1.65612\rangle$	
13	$\langle 0, P2, 1.79867\rangle$	2	1.79867		1.73458	2.32320	3.67351			$\langle 1, P1, 1.73458\rangle$	3
14	$\langle 0, P2, 1.79867\rangle$	2	1.79867	*	1.73458	2.32320	3.67351			$\langle 0, P1, 2.05182\rangle$	
15	$\langle 0, P2, 1.79867\rangle$	2	1.79867		1.73458	2.32320	3.67351			$\langle 0, P1, 2.05182\rangle$	
16	$\langle 1, P2, 1.79867\rangle$	3	1.79867		1.79867	2.05182	2.32320	3.67351		$\langle 0, P1, 2.05182\rangle$	

Figure 3: Sample trace of SE^{cons} for the small net.

can be passed on to following LPs. Hence the conservative strategy mainly relies on the possibility of having lookahead provided by the simulation model [Nico 88] [Lin 90].

If, for example, we follow the first simulation steps of a distributed implementation of SE^{cons} for our simple TTPN in Figure 3, we observe that the simulation engines in LP_1 and LP_2 synchronise each other to the earliest guaranteed *safe-to-process* times of transition firings. (The trace table in Figure 3 shows for LP$_1$ and LP$_2$ from left to right: the simulation step, the contents of the input buffer at the beginning of the step, the marking in the corresponding place, the local virtual time window LVTW, an indicator whether LP is blocked in that step, LVT, the transitions as scheduled for firing in the EVL, the contents of the outputbuffer at the end of the step, and the firing count FC for the transition.) In the first step both LPs are blocked due to their LVTW = 0.0. They send out nullmessages carrying the LP's lookahead computed as the sum of LVTW and the minimum random variate of the future

LP_1 on Processor 1							
S	IB	RB	P1	LVT_1	Eventlist $\langle T1@\ \rangle$	OB	FC
0			4	0.00000	0.70543 1.07530 1.34856 1.40297		
1			3	0.70543	1.07530 1.34856 1.40297	$\langle 1, P2, 0.70543, + \rangle$	1!
2			2	1.07530	1.34856 1.40297	$\langle 1, P2, 1.07530, + \rangle$	2
3	$\langle 1, P1, 0.83012, + \rangle$	*	3	0.70543	1.07530 1.34856 1.40297	$\langle 1, P2, 1.07530, - \rangle$	
			4	0.83012	1.07530 1.33622 1.34856 1.40297		
			3	1.07530	1.33622 1.34856 1.40297	$\langle 1, P2, 1.07530, + \rangle$	2!
4	$\langle 1, P1, 1.17333, + \rangle$		4	1.17333	1.33622 1.34856 1.40297 1.79867		
			3	1.33622	1.34856 1.40297 1.79867	$\langle 1, P2, 1.33622, + \rangle$	3!
5			2	1.34856	1.40297 1.79867	$\langle 1, P2, 1.34856, + \rangle$	4!
6	$\langle 1, P1, 3.67351, + \rangle$		1	1.40297	1.79867	$\langle 1, P2, 1.40297, + \rangle$	5
7	$\langle 1, P1, 3.67351, - \rangle$		0	1.79867		$\langle 1, P2, 1.79867, + \rangle$	
	$\langle 1, P1, 2.32320, + \rangle$						
8	$\langle 1, P1, 2.32320, - \rangle$		1	1.40297	1.79867	$\langle 1, P2, 1.79867, - \rangle$	5!
	$\langle 1, P1, 1.73458, + \rangle$	*	2	1.73458	1.79867 2.49959		
			1	1.79867	2.49959	$\langle 1, P2, 1.79867, + \rangle$	6!
9	$\langle 1, P1, 2.05182, + \rangle$		2	2.05182	2.49959 4.54256		
			1	2.49959	4.54256	$\langle 1, P2, 2.49959, + \rangle$	7
10	$\langle 1, P1, 2.32320, + \rangle$	*	2	2.05182	2.49959 4.54256	$\langle 1, P2, 2.49959, - \rangle$	
			3	2.32320	2.49959 3.02183 4.54256		
			2	2.49959	3.02183 4.54256	$\langle 1, P2, 2.49959, + \rangle$	7!
11	$\langle 1, P1, 3.13150, + \rangle$						

LP_2 on Processor 2							
S	IB	RB	P2	LVT_2	Eventlist $\langle T2@\ \rangle$	OB	FC
0			0	0.00000			
1			0	0.00000			
2	$\langle 1, P2, 0.70543, + \rangle$		1	0.70543	0.83012		
			0	0.83012		$\langle 1, P1, 0.83012, + \rangle$	1!
3	$\langle 1, P2, 1.07530, + \rangle$		1	1.07530	1.17333		
			0	1.17333		$\langle 1, P1, 1.17333, + \rangle$	2!
4	$\langle 1, P2, 1.07530, - \rangle$		0	1.17333			
	$\langle 1, P2, 1.07530, + \rangle$						
5	$\langle 1, P2, 1.33622, + \rangle$		1	1.33622	3.67351		
			0	3.67351		$\langle 1, P1, 3.67351, + \rangle$	3
6	$\langle 1, P2, 1.34856, + \rangle$	*	1	1.33622	3.67351	$\langle 1, P1, 3.67351, - \rangle$	
			2	1.34856	2.32320 3.67351		
			1	2.32320	3.67351	$\langle 1, P1, 2.32320, + \rangle$	5
7	$\langle 1, P2, 1.40297, + \rangle$	*	2	1.34856	2.32320 3.67351	$\langle 1, P1, 2.32320, - \rangle$	
			3	1.40297	1.73458 2.32320 3.67351		
			2	1.73458	2.32320 3.67351	$\langle 1, P1, 1.73458, + \rangle$	3!
8	$\langle 1, P2, 1.79867, + \rangle$		3	1.79867	2.05182 2.32320 3.67351		
			2	2.05182	2.32320 3.67351	$\langle 1, P1, 2.05182, + \rangle$	4!
9	$\langle 1, P2, 1.79867, - \rangle$		1	2.32320	3.67351	$\langle 1, P1, 2.32320, + \rangle$	5!
	$\langle 1, P2, 1.79867, + \rangle$						
10	$\langle 1, P2, 2.49959, + \rangle$		2	2.49959	3.13150 3.67351		
			1	3.13150	3.67351	$\langle 1, P1, 3.13150, + \rangle$	6
11	$\langle 1, P2, 2.49959, - \rangle$						
	$\langle 1, P2, 2.49959, + \rangle$						

Figure 4: Sample trace of SE^{opt} for the small net.

list (i.e., a set of n variates where n is the number of tokens in the P-invariant the transition is covered with; $n = 4$ in our example). Note that the first sending of nullmessages increases $LVTW_1$ from 0.0 to 0.09803 which is not wide enough to execute the internal event $\langle$T1@0.70543$\rangle$, hence both LPs remain blocked. In the second step LP_2 can 'improve' the timestamp of the nullmessage (whereas LP_1 cannot), now setting $LVTW_1$ to 0.80346 - thus permitting the occurrence of the first firing of T1 at time 0.70543, etc.

Although we have seen the potential parallelism in the trace in Figure 2, e.g. among the first occurrence of T2 and the second occurrence of T1, this model parallelism is not actually exploited by SE^{cons} to improve simulation performance in this particular example. In fact, whenever transition firing toggles from one LP to the other, three nullmessages are used to synchronise, forcing the partner LP to block for two steps (see this situation in Figure 3 at LP_1 steps 1–3, 4–7, 8–11, etc.)

3.2 Observations under Optimistic Simulation

An optimistic simulation engine SE^{opt} does not require external events (tokens) to appear in time order, the rollback mechanism takes care for proper synchronization respecting causalities. Rollback, however, requires a record of the LP's history with respect to the simulation of internal *and* external events. SE^{opt} hence has to keep sufficient internal state information in a *state stack* SS, allowing to restore a past state, and has to administrate an *input queue* IQ and an *output queue* OQ for storing message entries in increasing time stamp order. Since a time ordered arrival of token message cannot be guaranteed, either *positive* (token) messages ($m = \langle \#, P, TT, \text{`}+\text{'} \rangle$) or *negative (anti-) messages* ($m = \langle \#, P, TT, \text{`}-\text{'} \rangle$) are exchanged among LPs indicating either the transfer of a token or the request to annihilate a previously received positive message. Obviously, the communication overhead caused by antimessages (rollback) imposes a dramatic burden on the simulation performance of a DDES running on a multiprocessor.

Several strategies have been developed at least in theory to improve *absolute*, but more often *relative* distributed/parallel simulation performance. In the original Time Warp [Jeff 85a] [Jeff 85b] an LP receiving a message with timestamp less than LVT (*straggler message*) initiates sending antimessages immediately (*aggressive cancellation*, *ac*) when executing the rollback procedure. Since the impact of antimessages might be that also succeeding LPs are forced to roll back, a *chain of rollbacks* that eventually terminates is generated. Reducing the size of the rollback chain can be attempted by "filtering" messages with preempted effects, [Prak 91], by postponing erroneous message annihilation until it turns out that they are not reproduced in the repeated simulation (*lazy cancellation*, *lc*), [Gafn 88], or by maintaining a causality record for each event to support *direct cancellation* of messages that will definitely not be reproduced [Kona 91].

The traces in Figure 4 represent the behavior of the optimistic protocol with the *lc* message annihilation strategy (the trace table is to be read in the same way as the one in Figure 3, except that there is a rollback indicator colum RB instead of a blocking column B; an entry in FC without ! denotes a firing of a transition that could be rolled back eventually, whereas one with ! denotes a confirmed firing):

We see that in contrast to SE^{cons} LP_1 can start immediately simulating internal events, producing $\langle 1, P2, 0.70543, + \rangle$ for LP_1 in the first step. In the second step the events $\langle T1@1.07530 \rangle$ and $\langle T2@0.83012 \rangle$ are simulated in parallel by the two LPs. In the same step LP_1 sends $\langle 1, P2, 1.07530, + \rangle$, while LP_2 sends $\langle 1, P1, 0.83012, + \rangle$, which is beeing considered a straggler by LP_1 (timestamp of message $<$ LVT), forcing it to rollback to the beginning of step 3. Rollback means, first, restoring the most recent state, the one after step 1, and second, sending out an annihilation message $\langle 1, P2, 1.07530, - \rangle$ for the positive message sent in step 2. The *ac* policy would send out that anti-message immediately, such that LP_2 would find it in IB at the beginning of step 4; but at that time LP_2 would already have produced (and sent!) a new message to LP_1, namely $\langle 1, P1, 1.17333, + \rangle$. Recognizing the antimessage in step 4 would hence force LP_2 to roll back in turn, thus annihilating what has been sent in step 3, etc. This situation is called a rollback cascade (or chain) that would break eventually, but a tremendous amount of communication overhead would have been produced compared to the computational efforts for simulating an internal event.

As mentioned, the *lc* policy improves the communication overhead caused by these 'unnecessary' rollback chains by delaying the sending of antimessages: Step 3 of LP_1 shows that resimulation after rollback produces exactly the same positive message ($\langle 1, P2, 1.07530, + \rangle$) that was sent in step 2, and which was ment to become annihilated at the beginning of step 3. As an improvement, *lc* annihilates the dual messages locally, without sending any message at all, thus breaking the rollback chain earlier than *ac*.

3.3 Motivation for a Probabilistic Simulation

3.3.1 Theoretical Motivation

Although the *lc* policy considerably improves absolute simulation performance of *ac*, it is still not an optimum synchronization protocol: Given that every LP in every step knew exactly which messages (timestamps) are supposed to arrive, it could easily adjust its local behavior in order to avoid *any* rollback. Reducing the number of synchronization messages by avoiding rollbacks as far as possible is the motivation of the probabilistic protocol.

To illustrate the idea by an example, look again at Figure 4: Up until step 6 LP_2 has collected the following vector of token time observations $O = (0.70543, 1.07530, 1.33622)$, i.e. the token time increments (TTI) 0.36987 and 0.26092. Arguing with an 'average' TTI of 0.315395, LP_2 would expect the next (positive) message to have token time at around 1.33622 + TTI = 1.651615. If LP_2 would have used this information in step 6 in the sense of awaiting the next token instead of progressing LVT to 2.32320 by firing one transition, the message $\langle 1, P1, 2.32320, + \rangle$ would not have been produced and sent. In fact, at the beginning of step 7 a new message $\langle 1, P2, 1.40297, + \rangle$ close to the estimate 1.651615 arrives, forcing LP_2 to rollback step 6, and, what is very performance inefficient in a distributed memory environment, to send an antimessage: $\langle 1, P1, 2.32320, - \rangle$. The probabilistic simulation protocol, by making use of next token time estimates, would have avoided the two basically unnecessary messages.

Awaiting the next token can be implemented by stoping the local event simulation (advancing LVT) at the estimated next token time (NTT). This policy, however, leads to deadlock caused by cyclic waiting conditions imposed by the underestimation of NTT of all LPs in a directed cycle. In our protocol therefore, the SE does not stop definitely when reaching the NTT estimate, but only with some probability that is related to the *confidence* in the prediction (which would be low in the context of the previous paragraph)

3.3.2 Practical Motivation

Both the optimsitic and the conservative protocol have been implemented on a variety of parallel machines (Transputer, Sequent Balance B21 and Intel iPSC/860). The following performance observations were made when simulating the TTPN in Figure 3 on the iPSC (values averaged over 30 independent runs).

Conservative Simulation on the iPSC/860 A first glance at Figure 5, showing the absolute number of events processed (external and internal) (left), and the proportion of time that SE^{cons} spent for event processing, sending/receiving messages and waiting for messages (right, both for LP1), shows that communication latency is dominating the processing power for the particular TTPN. Controlling the degree of model parallelism with token duplication in place P1 causes an exponential increase of nullmessages necessary for synchronization. The computational requirements for 'firing one transition' are almost neglectible (a couple of processor cycles on a 40 MHz, 80 MFLOPS peak rated i860 processor), compared to sending messages (the current implementation of the protocol uses 10 Byte messages, such that communication latency between two adjacent nodes on the iPSC/860 using synchronous message passing is around 75μsec ($T_{comm}(B) = 69\mu\text{sec} + \text{No\_of\_BYTES} * 0.43\mu\text{sec}$)). A radical reduction of communication overhead – or in other words, an increment of local event processing between synchronization – is necessary in order to achieve real speedup over the sequential simulation. The blocking restriciton of SE^{cons} *has to be* weakened, also from the implementation (on contemporary parallel platforms) point of view. Note that due to symmetry ($\lambda_1 = 1.0, \lambda_2 = 1.0$) the behavior of LP_2 is exactly the same as for LP_1.

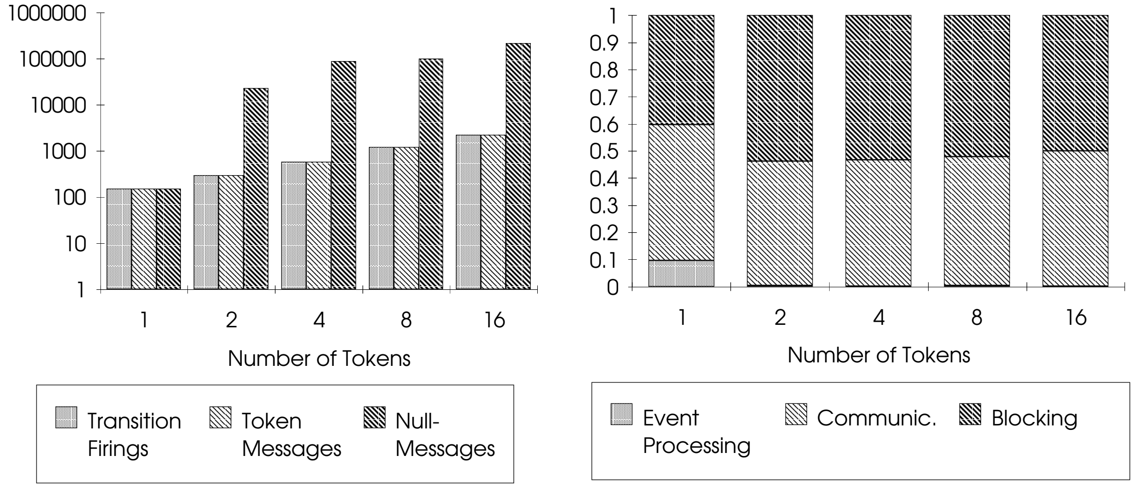

Figure 5: SE^{cons} Performance on the iPSC/860 Hypercube

Optimistic Simulation on the iPSC/860 Communication dominance is also evident from performance measurement of SE^{opt} on the iPSC, which is documented in Figure 6 and 7. Besides variations of the model parallelism by duplicating tokens in the initial marking, we here also present variations of the 'balance' of the TTI in both LPs and their impact on the performance in order to demonstrate the sensitivity of SE^{opt} with respect to imbalance. Balance is controlled by λ_2; with $\lambda_2 = 1.0 (= \lambda_1)$ we have the balanced situation of Figure 3. Setting $\lambda_2 = 1/2$ makes the TTI of LP_2 twice as high than the one in LP_1, i.e. the enabling time of T2 is on average twice as long; the higher the expected firing delay of T2, which is $\frac{1}{\lambda_2}$, the more of the available tokens will reside in P2 (in steady state) enabling T2.

Figure 6 shows balance and model parallelism impact onto the number of rollbacks and antimessages in the optimistic protocol with lazy cancellation. With one token there is no model parallelism at all, no rollbacks/antimessages can occur. Event processing speed (i.e. number of transitions fired during the event processing phase) is high compared to higher number of tokens, since EVL can have one entry at the maximum and consequently the EVL management and state saving is also relatively fast. The proportion of time spent waiting to receive a message (blocking time ratio) and of sending/receiving messages (communication time ratio) is balanced (0.53, 0.43), but high compared to the proportion of time spent for firing transitions (event processing time ratio) (0.4) (see Figure 7).

In the balanced situation $\lambda_2 = 1$ the percentage of rollback steps out of simulation steps (transition firings) is decreasing with the amount of model parallelism. Consequently the EVL management and state saving efforts are more complex, increasing the event processing time ratio, i.e. slowing down (event processing) speed of the LPs. Since there is higher chance of actual parallelism exploitation when there are more tokens in the net, blocking goes down.

Consider on the other hand the model parallelism fixed to 4 tokens, and a varying imbalance parameter λ_2. Since decreasing λ_2 causes a shift of tokens from P1 to P2 in the steady state sense (with $\lambda_2 = \frac{1}{16}$ the expected number of tokens in P2 is 3.7647 [1] (out of 4), whereas it is 2 in the case of $\lambda_2 = 1$), EVL management and state saving costs increase (decrease) in LP_2 (LP_1) – the event processing speed of LP_2 (LP_1) slows down (improves) (Figure 6). Blocking time has the same, communication time the opposite tendency. Moreover, the more model parallelism we have, the more

[1] The continuous time markov chain (CTMC) underlying the TTPN has five states, i.e $s_1 = (4, 0)$ (denoting 4 tokens in P1 and no token in P2), $s_2 = (3, 1)$, etc.. Solving for the steady state probability vector $\vec{\pi} Q = \vec{0}$ where Q is the infinitesimal generator of the CTMC yields $\vec{\pi} = (0.00001, 0.00077, 0.01839, 0.19617, 0.78466)$, and therefore $E[\text{tokens\_in\_p2}] = 3.7647$.

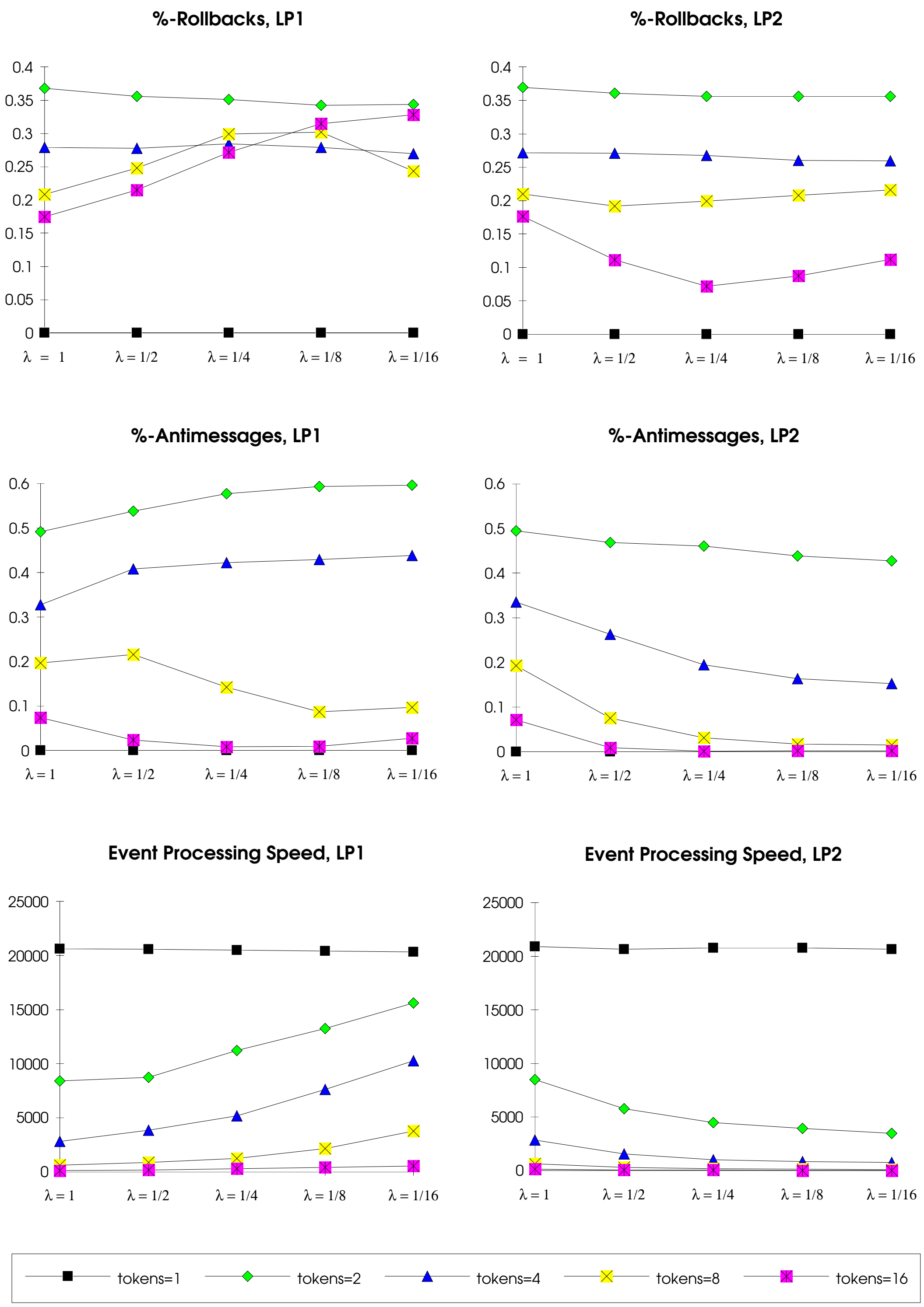

Figure 6: SE^{opt_lc} Performance for LP$_1$ on the iPSC/860 Hypercube

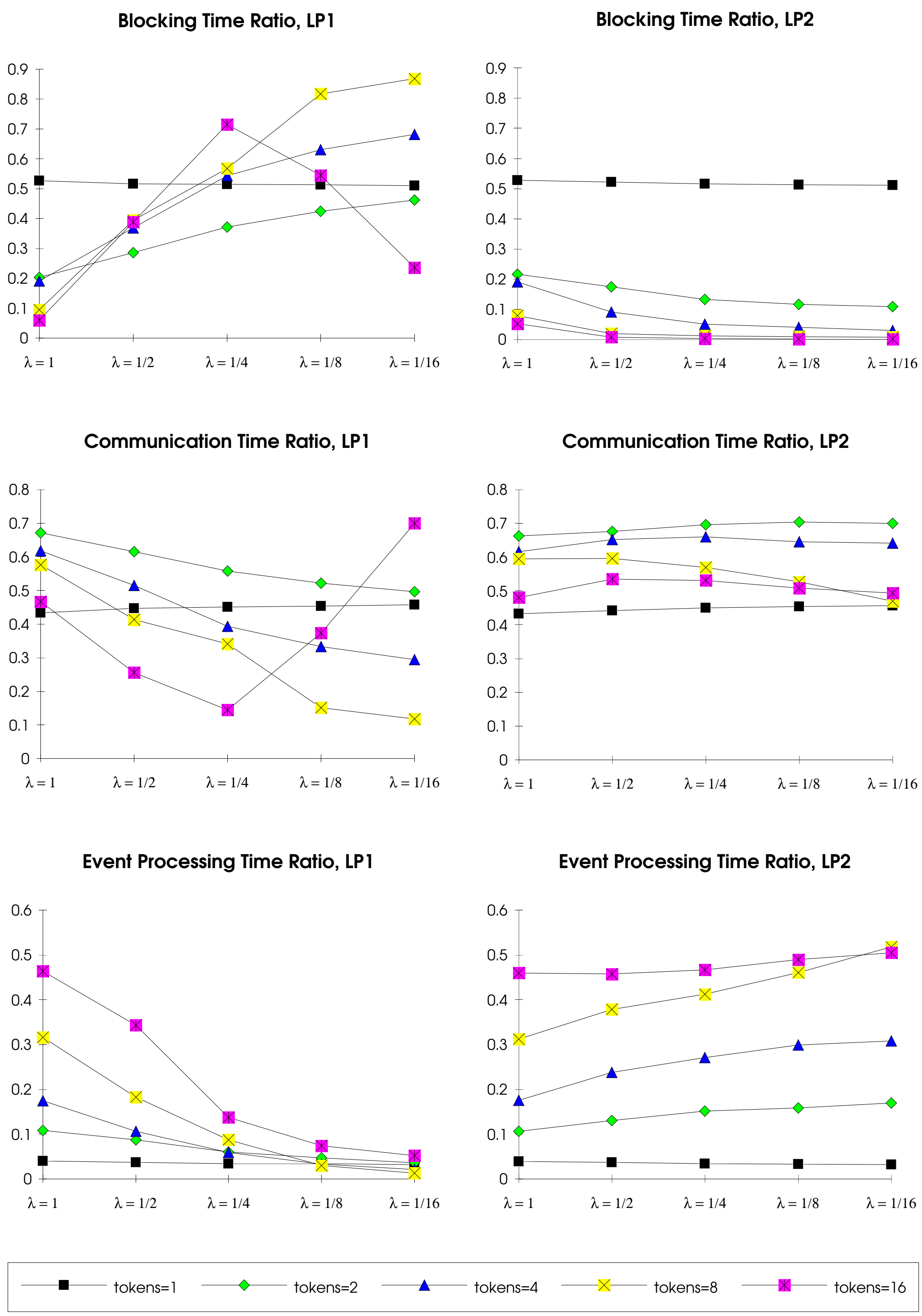

Figure 7: SE^{opt_lc} Execution Time on the iPSC/860 Hypercube

significant are these tendencies.

With respect to rollback we can now conclude, that with increasing model parallelism event processing speed is reduced. The more imbalanced the timing in the model, the more load moves to LP_2, thus speeding up LP_1. This observation is in conformance with the arguments we have drawn from the underlying CTMC. This load imbalance is in fact responsible for the chance of having rollback in a step (Figure 6). In LP_2, which increments LVT 16 times as fast as LP_1 in case of $\lambda_2 = \frac{1}{16}$, approximately every third step is a rollback step when 2 tokens are circulating. Having 16 tokens in the TTPN, approx. every tenth step is a rollback step. We observe almost the opposite in LP_1, since it is the 'faster' component. Fortunately (and obviously), the lazy cancellation strategy finds higher chance for local antimessage annihilation, thus reducing the percentage of antimessages sent despite increasing percentage of rollbacks. This has also been seen from the small sample trace in the previous chapter for *lc*. Again the improvement by *lc* indicates that most of the rollbacks in LP_1 are 'unnecessary' and could have been avoided by slowing down LP_1 using probabilistic blocking for awaiting the next token. We see, that also from a practical implementation point of view, the expected effects of the probabilistic protocol are promising.

4 The Probabilistic Protocol

4.1 Informal Description

The basic idea of improving the optimistic scheme has been seen by [Soko 88] in the Moving Time Window (MTW) approach, which uses a fixed size time window W such that only events e_i with occurrence time $\mathrm{LVT} \leq occ(e_i) \leq \mathrm{LVT} + \mathrm{W}$ are simulated optimistically, thus preventing the simulator from incorrect computations to be propagated too far ahead into the simulated future. Our approach could be interpreted as a time windowing scheme with *dynamical* changes in the size of W, related to the inherent model parallelism. (We will call the time instant at which there is (probabilistic) reason to assume an external event to arrive as the *local virtual time window* (LVTW).) Our protocol does not strictly adhere the W, but with some probability related to the confidence in the estimate of LVTW. The probabilistic scheme to be described in the sequel adapts the size of W according to the observed arrival process of external events. It performs a collection of arrival observations in order to provide a reasonable estimate for LVTW 'on-the-fly', i.e. with neglectible intrusion to the simulation engine.

4.2 Pseudocode and Data Structures

The operational behavior of a logical process simulation implemented using the probabilistic protocol is now described in terms of the data structures (see Figures 8 and 9) and the synchronization mechanism employed by every $LP_k = (R_k, I_k, SE)$. Given the partitioning of a TTPN into regions the internal representation of R_k is straightforward and will not be described here.

The communication interface I_k of some LP_k is dedicated to handle the distribution and collection of *packets* on its incoming and outgoing channels. For every '*input*' channel $ch_{i,k} \in CH$ directed towards LP_k an *input buffer* IB[i] is used for depositing arriving packets. When a message from LP_k has to be sent to some LP_j via $ch_{k,j} \in CH$, it is 'packed' into a packet and deposited into an *output buffer* OB[j] before actual sending is invoked. LP_k uses one OB[i] for every output channel. Information that is related to synchronization constraints imposed by *packets* arriving at channel $ch_{i,k}$ is kept in a ddata structure called *channel information* in CH[i]. Related to the protocol internal data structures are necessary to maintain the history of yet uncommitted external events, i.e. token messages that might be subject to annihilation due to occasional rollbacks in an *input queue* IQ, and already sent token messages that might have to be invalidated due to a local rollback in an *output queue* OQ. A *rollback queue* RQ serves as a temporary deposit of antimessages until it is confirmed that a certain antimessage cannot be annihilated locally due to the recomputation of a past state that caused the original (positive) token message. All the processing of events is logged into a *state stack* SS in terms of time/state tuples to allow a post mortem analysis of the simulated (physical) system.

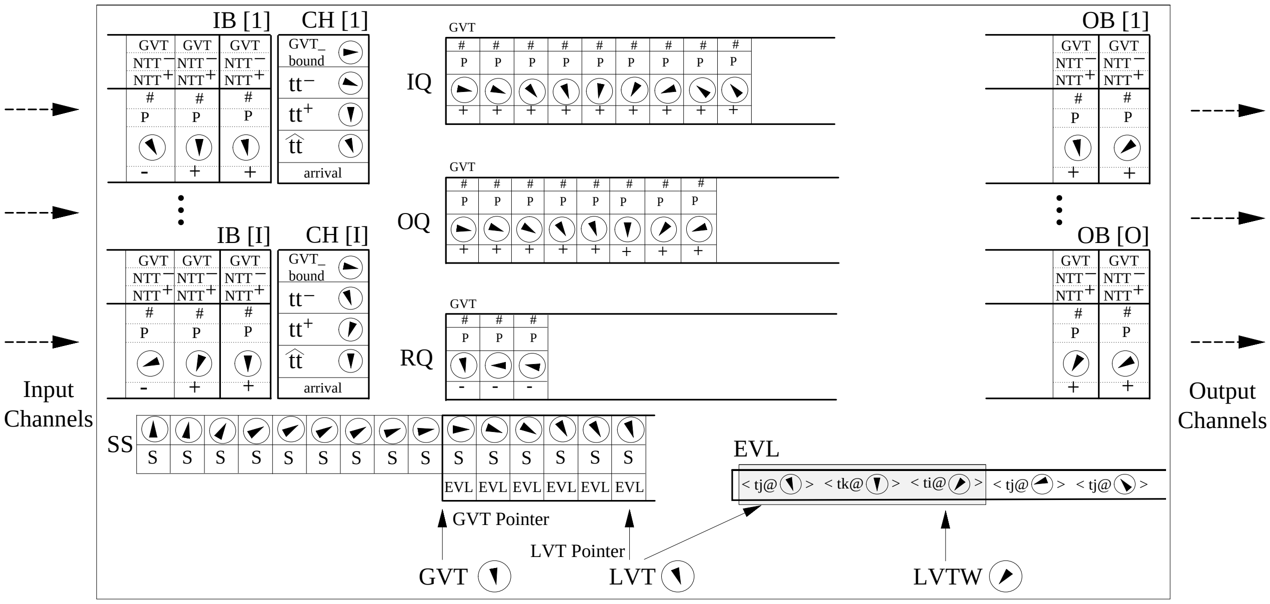

Figure 8: Logical Process for the Probabilistic Protocol

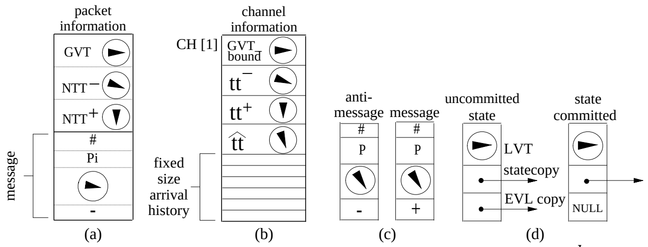

Figure 9: Data Structures for SE^{prob}.

For a state generated in one simulation step for which no guarantee exists that it could become invalid eventually, i.e. the event time is between the LVT and the GVT, a copy of the *event list* EVL also has to be stacked, in order to be able to restore a past state on the occurrence of rollback.

The simulation engine SE^{prob} for an R_k operates now as follows (see Figure 10 for a detailed algorithm description): After an initialization of the data structures and clocks, the region R_k is read into LP_k and for each transition with random firing delay a set of random numbers is presampled. According to these future lists and the initial marking in that region, enabled transitions are scheduled by inserting event entries into EVL, respecting increasing occurrence time order. This initial state is pushed on SS, and the lookahead imposed on the OBs by the inital state is determined (end of step *S1*). After that the algorithm is entering a loop, where first SE^{prob} looks at all IBs for new packet arrivals (see Figure 9 (a)). A packet carries two different kinds of information: 1) information that is related to the characterization of the arrivals and the synchronization constraints induced by the preceding LP (type 1), and 2) information on external events coming from that LP. Consider for first only the external event information, which is separated from an arriving packet as the *message* contained in that packet. The timestamp of that message (external event) is used to find the straggler with minimum timestamp among all stragglers observed over all packets in all IBs (step *S2.1*). Given that a straggler was found at all, then step *S2.2* invokes a rollback to the timestamp of the straggler, by moving the copies of intentionally sent messages with timestamp higher then the one of the straggler from OQ to RQ, while simultaneously converting their sign. Keeping this list of antimessages is done for the chance of local annihilation (lazy cancellation), which can be decided for every antimessage only after proceeding the simulation (LVT) over its timestamp. After that, packets that have been deposited in the OBs in the previous iteration of the loop are sent out to other LPs in *S2.5*. (Leave out *S2.3*, *S2.4* and *S2.7* for the moment since the information processed there is again of type 1).) In *S2.6* it is decided whether the stopping criterion holds or not. If not, SE^{prob} determines whether to simulate an internal (*S2.9*) or an external event (*S2.10*) in the next step, depending on their respective timestamps. Simulating an internal event means to execute the effect of the internal event (in our case a transition firing along with the scheduling of new transitions and the descheduling of obsolete transitions, and the collection of tokens produced by the firing and to be deposited in places at some other R_j in a list of preliminary outputmessages), while trying the chance of local annihilation. This is done by finding (and cancelling) dual messages in the ddata structure *outmessagelist* and RQ, while moving the remaining rest of RQ into the outmessagelist. The simulation of an external event (*S2.10*) means to change the marking in R_k according to the number of tokens delivered by the *nextmessage* (in IQ) into the specified places. Both internal and external events increment LVT, and cause a new state of LP_k. Since the event as processed for now is not committed, sufficient information has to be stored to retrieve the current state on the occasion of rollback. In order for that, SE^{prob} stores (*S2.11*) – along with LVT – a copy of the generic state information in the *statecopy* component of a state template (see Figure 9 (*d*)), as well as a copy of the current EVL. The latter has to be stored only for uncommitted states and will be purged as soon as GVT sweeps past the timestamp of the state (*S2.3*). The new state reached imposes new lookahead information on the OBs, which is collected in *S2.12* to help determining bounds on the next token message to be sent via an output channel. This additional information is packed together with the messages in outmessagelist to form a new packet, which is preliminarily deposited in the corresponding OB[i] (*S 2.13*), and sent out at the beginning of next loop iteration. At this point one can see why the sendout of packets is placed on top of the loop: since while SE^{prob} is simulating steps *S2.7* to *S2.13*, new messages might have arrived in the IBs, occasionally causing rollback. To prevent us now from sending packets that could already be obsolete, we first look at the IBs for stragglers and perform rollback before sending out the packets – thus exploiting one more chance of early rollback chain breaking. It is now clear, that 'try_local_annihilation' also has to look into the OBs when looking for messages dual to the ones in RQ as stated in *S2.9*.

GVT Computation Since [Jeff 85a] GVT has been used to refer to the lower bound on LVT to which any LP_k will ever roll back, and algorithms for a distributed computation of GVT have been developed since [Sama 85] (see also [Bell 90]). We have developed a GVT algorithm that does not impose additional communication complexity on the protocol, rather than e.g. a distributed snapshot algorithm [Ben 90] would. SE^{prot} computes GVT by looking at the GVTs provided by preceding LPs. Every packet received via some input channel $ch_{i,k}$ comprises as its first component the GVT of LP_i. When reading the packet out of the IB[i] in *S2.1*, the GVT_bound of channel CH[i] is stored by the update_channel procedure, which is detailed in Figure 11: In *UC1* the GVT that was received with the packet is stored in IC[i], and if this is an improvement to LP_k's GVT, this is adjusted (*UC2*). On the other hand, LP_k provides its 'local' GVT value to all LP_j's, by attaching GVT incremented by the lookahead imposed by the current state on the corresponding output channel OB[j] to every packet that will be sent via output channel j (see *S2.5* in Figure 10). Since the token messages would be sent anyhow, the intrusion of our GVT computation is just extending the message (packet) size by one data item of type 'TIME', and the local computation of lookahead for the last committed state (step *S2.4*). Obviously GVT – representing the set of committed events – is used to terminate SE_k^{prob} in steps *S2.6*.

Next Token Time and the Local Virtual Time Window To support the next LPs for their estimation of the arrival of the next external event, LP_k sends out on each of its output channels bounds on the *next token time* (NTT). The lower bound on the timestamp of the next (external) token produced by SE_k^{prob} for LP_j is exactly the lookahead of the state (SE_k^{prob} is in) imposed on OB[j]. A corresponding upper bound can be computed from the precomputed random lists in certain cases,

```
program SE^prob (R_k) {

S1      GVT=LVT=LVTW=0.0; IQ=OQ=RQ=NULL; SS=NULL;
        initialize(R_k); initial_state = generate_initial_state();
        for each IC[i] do set_initial_bounds(IC[i]);
        LVTpointer=GVTpointer=push_state(inital_state, &SS);
            for each OB[i] do LVT_lookahead[i] = getlookahead(LVTpointer, OB[i]);
        LVT_lookahead = lookahead(LVTpointer);
S2      do {
            loop:
            straggler=NULL; GVT=∞;
S2.1        for each IB[i] do
                for each packet[j] in IB[i] do {
                    update_channel(IC[i].arrival, packet[j]);
                    message = unpack(packet[j]);
                    if (message→timestamp < LVT) straggler = adjust_straggler(message);
                    move_to_IQ(message); remove(packet[j], IB[i]);
                }
S2.2        if (straggler != NULL) { /* rollback */
                LVTpointer=rollback(straggler→timestamp, OQ, RQ);
                restore_state(LVTpointer); LVT = LVTpointer→Lvt;
            }
S2.3        GVTpointer = release_memory(GVT);
S2.4        for each OB[i] do GVT_lookahead[i] = getlookahead(GVTpointer, OB[i]);
S2.5        for each OB[i] do /* sendout packets in OB */
                for each packet[j] ∈ OB[i] do {
                    OB[i].packet[j].GVT = GVT + GVT_lookahead[i];
                    store_message(packet[j], OQ); remove(packet[j], OB[i]);
                }
            }
S2.6        stop = (GVT > ENDTIME); if (stop) break;
S2.7        for each IC[i] do IC[i].t̂t, confidence[i] = estimate(IC[i].arrival, IC[i].tt⁺, IC[i].tt⁻);
            i = min(IC[ ].t̂t); LVTW = IC[i].t̂t; confidence = confidence[i];
            /* is there something to do? */
            nexteventtime = gettime_first(EVL); /* ∞ if empty */
S2.8        if (nexteventime >= LVTW) {
                if (rand_uniform(seed) <= confidence) goto loop;
            }
            nextmessage = getfirst(IQ, LVT); nextmessagetime = gettimefirst(IQ, LVT); /* ∞ if empty */
            outmessagelist = NULL;
S2.9        if (nexteventtime < nextmessagetime) { /* simulate first internal event */
                LVT = nexteventtime;
                outmessagelist = firefirst_EVL();
                try_local_annihilation(RQ, OB[ ], outmessagelist);
                move_rest(RQ, outmessagelist);
            }
S2.10       else { /* simulate first external event */
                LVT = nextmessagetime;
                changemarking(nextmessage);
            }
S2.11       state = generate_newstate(LVT, getcopy_State(), getcopy_EVL());
            push_state(state, &SS); LVTpointer = state;
S2.12       for each OB[i] do LVT_lookahead[i] = getlookahead(LVTpointer, OB[i]);
            for each message ∈ outmessagelist do {
S2.13           packet = generate_packet(message); outbuffer = which_LP(message);
                packet→NTT⁻ = LVT + LVT_lookahaead[outbuffer];
                packet→NTT⁺ = get_NTT_upper_bound(outbuffer);
                remove(message, outmessagelist); insert(packet, OB[outbuffer]);
            }
        }
}
```

Figure 10: Simulation Engine for a TTPN Region R_k.

```
program update_channel (IC, packet) {
UC1    IC.GVT_bound = packet.GVT
UC2    GVT = adjustGVT(GVT, IC.GVT_bound);
UC3    IC.tt+ = packet.NTT+;
UC4    IC.tt- = packet.NTT-;
UC5    IC.tt^ = UNDEFINED;
UC6    update_arrivalhistory(IC.arrival, packet);
       }
```

Figure 11: Update of Channel Information upon Receipt of a Message.

where structural (pre-)analysis of the TTPN model revealed synchronic distances among transitions. These two timevalues are added to every message when packets are constructed for each message in the outmessagelist (*S2.13*). Upon receipt of a packet (*S2.1*) this information is extracted and kept in the corresponding CHs of IBs as detailed in Figure 11, steps *UC3* and *UC4*. The estimate of the next external event time IC.$\widehat{tt}$ to be observed on that channel is intentionally set to be undefined (*UC5*), and will be set right before the simulation of an (internal or external) event in *S2.7* (the estimation as such will be worked out in detail in the next section). The estimation is based on an observation of the arrival process for a certain amount of messages received in the past. The arrival history as used by *S2.7* is also maintained upon packet receipts (Figure 11, *UC6*). The minimum of estimates IC[i].$\widehat{tt}$ determines LVTW.

The estimation procedure in *S2.7* also returns a confidence value normalized to the scale $(0, 1)$, expressing statistical evidence for IC[i].$\widehat{tt}$. The probabilistic blocking is now implemented in *S2.8*: first it is checked whether the next scheduled event is before LVTW. If so, simulation proceeds. If not, the SE blocks with a 'confidence-%' chance by branching back to the main loop.

5 Characterization of the Arrival Process

5.1 Next Token Time Point Estimates

Mean Token Time Increment An obvious approach of characterizing the arrival process via the input channel IC[i] in order to estimate the next token time IC[i].$\widehat{tt}$ (in further just $\widehat{tt}$) for that channel (as required by the probabilistic protocol in Figure 10) could be by determining the average and variability of channel time increments based on the observed arrival history IC[i].arrival. Let IC[i].arrival $= (tt_0, tt_1, tt_2, \dots tt_n)$, $tt_{i-1} \leq tt_i, i = 1, \dots n$ be the tokentime observation in a history window of size n, and $O = (\delta_1, \delta_2, \dots \delta_n)$ be the observation vector of observed token time increments $\delta_i = (tt_i - tt_{i-1})$ (TTI). Then $\widehat{tt}$ can be estimated to be $tt_n + \widehat{\Delta}$ where the observed mean $\widehat{\Delta} = \overline{\delta} = \frac{1}{n}\sum_{i=1}^{n} \delta_i$ (but in any case $tt^- \leq \widehat{tt} \leq tt^+$) could be used as an estimate with some confidence $\Phi = 1 - scale(s/\widehat{\Delta})$ based on the scaled coefficient of variation ($s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\delta_i - \widehat{\Delta})^2}$). In general, for the arrival process of tokens in an arbitrarily partitioned TTPN the observed sequence of tokentimes cannot be assumed to follow an unimodal, symmetrical distribution, hence more care has to be devoted when selecting the index of dispersion for the confidence value. For a skewed or multimodal probability distribution function (pdf) the confidence value should be based on a level-β semi-interquantile range $\frac{\delta_{(1-\beta)} - \delta_{(\beta)}}{2}$, where $\delta_{(1-\beta)}$ ($\delta_{(\beta)}$) is the $(1-\beta)$ (β) quantile, e.g. $\Phi = \frac{\delta_{(1-\beta)} - \delta_{(\beta)}}{2}$.

The major advantage of this approach is its trivial implementation, and the fact that $\widehat{\Delta}$ can be computed *incrementally* without storing the sequence of observations: $\widehat{\Delta}_{n+1} = \frac{n\widehat{\Delta}_n + \delta_{n+1}}{n+1}$. The evaluation of this expression is a completely neglectible source of potential intrusion once Φ has been fixed to a constant value.

(Curve-) Linear Regression Another way of providing an estimate of the next token time would be using the response variable $\widehat{\Delta}$ of a (curve-) linear regression model $\widehat{\Delta} = b_0 + b_1 x$ for the next predictor x. (The regression parameters giving minimum error variance are obtained as $b_1 = \frac{\sum x_i\delta_i - (\frac{1}{n}\sum x_i)(\frac{1}{n}\sum \delta_i)}{\sum x_i^2 - n(\frac{1}{n}\sum x_i)^2}$

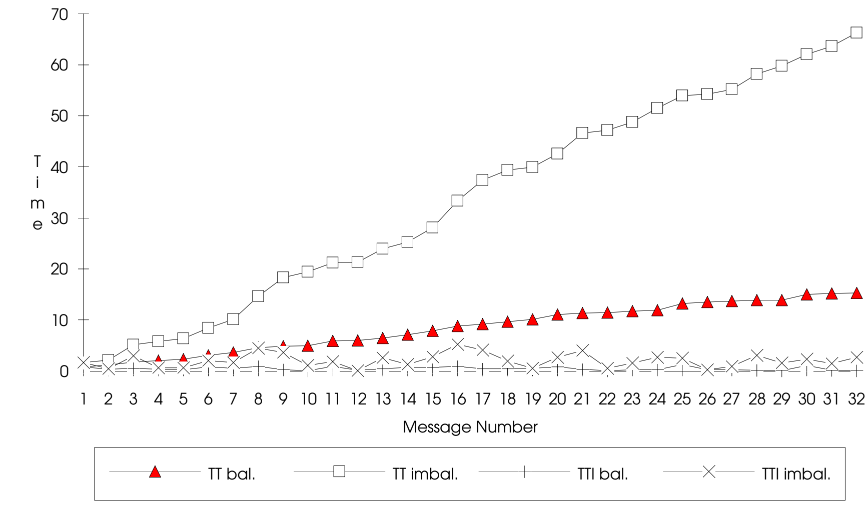

Figure 12: Arrival Process as observed by LP_1

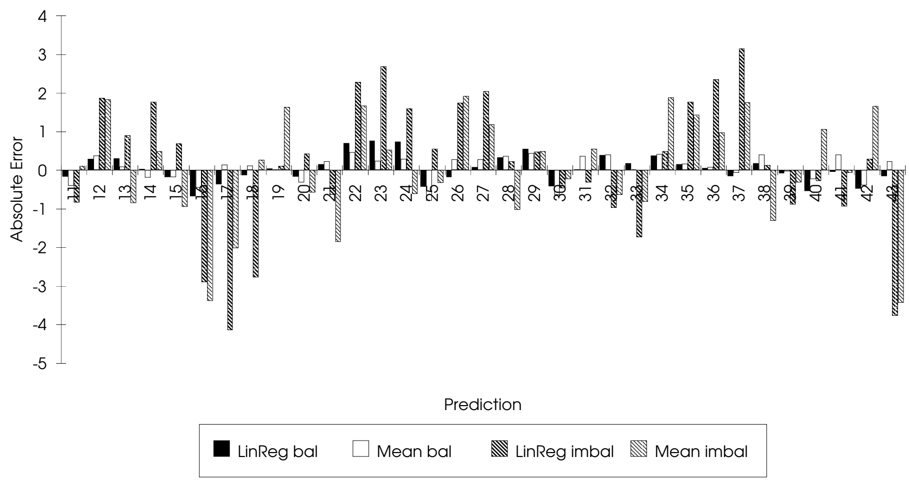

Figure 13: Predicition Error of Linear Regression and Mean in LP_1

and $b_0 = \frac{1}{n}\sum \delta_i - b_1(\frac{1}{n}\sum x_i)$ from the arrival history O.) The confidence for $\widehat{\Delta}$ would be expressed in terms of the coefficient of determination as $\Phi = R^2$, where $R^2 = \frac{SST-SSE}{SST}$, SST being the total sum of squares and SSE the sum of squared errors. The confidence in the regression parameters b_0 and b_1 at level β could be considered to adjust Φ, related to confidence intervals $b_0 \pm t_{[\frac{1-\beta}{2};n-2]} s_{b_0}$ and $b_1 \pm t_{[\frac{1-\beta}{2};n-2]} s_{b_1}$, where $t_{[\frac{1-\beta}{2};n-2]}$ is the $\frac{1-\beta}{2}$ quantile of a student t variate with $n-2$ degrees of freedom, and s_{b_0} (s_{b_1}) is the standard deviation of the error. If the relationship among the response $\widehat{\Delta}$ and the predictor x appears to be nonlinear, there might still be some hope to be able to convert to linear form (e.g. if the relationship is exponential ($\widehat{\Delta} = ax^b$) then $ln(\widehat{\Delta}) = ln(a) + b\ ln(x)$ is linear), otherwise a nonlinear regression model would have to be applied.

Generally, the arrival process of tokens does not give rise for any particular assumption of the regression model – only for trivial cases of LP behavior linear assumption is a feasible choice. Figure 13 shows the application of a linear regression model (observation window $n = 10$) (compared to the previous 'mean' model) to the arrival process of tokens as observed by LP_1 in our example (4 tokens, $\lambda_2 = 1$ is the balanced case, $\lambda_2 = \frac{1}{4}$ the imbalanced; Figure 12 shows a sample path of the arrival process as observed by LP_1 (TT stands for the tokentime of the message, TTI is the tokentime increment caused by the most recent message), which was used for the prediction in Figure 13). We have shown with several experiments not documented here, that linear regression models do *not* perform significantly better than the 'mean' model – although their computational and memory

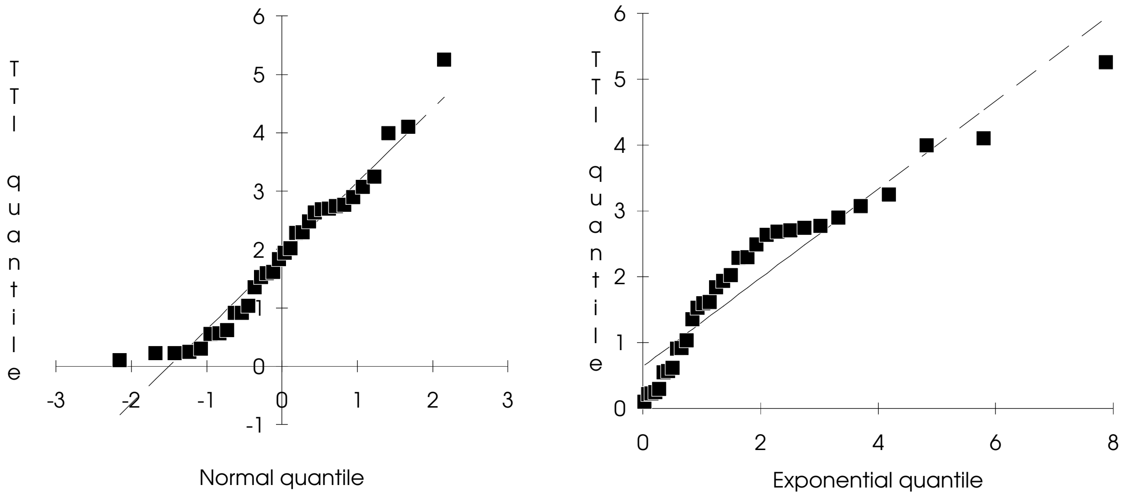

Figure 14: (a) $Q^2_{N(0,1)}$ and (b) $Q^2_{exp(\lambda)}$ for an $n = 2^5$ prediction window at LP_1.

complexity is tremendously higher. Thus, for practical implementations, linear regression models can be considered obsolete.

5.2 Assuming an Underlying pdf

In order to improve the $\widehat{tt}$ compared to the consideration of the variability and dispersion of the history in the previous section, we can state on the type of the probability distribution function (pdf) that underlies the sample. The central limit theorem, saying that the sum of a large number of *independent* observations from any distribution tends to be $N(\mu, \sigma^2)$ distributed, cannot be assumed to hold due the lack of evidence for independence. So, at least for our purpose, assumptions for a specific pdf are essential.

Given some hypothesis test rises evidence for assuming that the sample follows some theoretical pdf (e.g. exponential, normal), then $\widehat{tt}$ can be determined by drawing a random variate according to that pdf. We have experimented with hypothesis tests like the Kolmogoroff-Smirnov test and the χ^2-distribution test, and again found comparably high computational complexity, which we tried to eliminate by exploiting the hypothesis-testing potentials of quantile-quantile-plots.

Assume the sequence of TTIs $O = (\delta_1, \delta_2 \ldots \delta_n)$ for a prediction window of $n = 2^d$ (for practical application $5 \leq d \leq 8$), and a set of theoretical distributions $F_i(x)$, $i = 1, 2, \ldots N$. In order to find the maximum likelihood $F_k(x)$ for the observation sequence O we investigate the function $Q^2_{F_k}(x_i, \delta_{(i)})$, relating the q_i-th *observed* quantile $\delta_{(i)}$ to the q_i-th quantile x_i coming from the *theoretical* CDF (Cumulative Distribution Function) $F_k(x)$. So for example let $F(x) = 1 - e^{-\lambda x}$ be the CDF of the exponential distribution, then the q_i-th quantile ($q_i = [\frac{i-0.5}{n}]$) of $F(x)$ is $x_i = F^{-1}(q_i) = -(\frac{1}{\lambda}) ln(1-q_i)$. The q_i-th quantile $\delta_{(i)}$ of the observations is found as the i-th element in the ordered sequence of observations. Obviously the n plot points of Q^2 can be generated for CDF's for which a closed form for $F^{-1}(x)$ exists. In all other cases the sequence of n quantiles can be derived from tables or approximations. In any case the quantiles x_i are precomputed for all the $F_k(x)$'s and stored within the LPs (as an $n \times N$ quantile matrix).

The $F_k(x)$ of choice is the one for which $SSE(Q^2_{F_k}(x_i, \delta_{(i)}), LSLF(Q^2_{F_k}(x_i, \delta_{(i)})))$, the *sum of squared errors* among $Q^2_{F_k}$ and its *least square linear fit* (LSLF), is minimum over all $k = 1, 2, \ldots N$.

The Q^2 plots in Figure 14 compare the fitting of an $n = 32$ sequence of TTIs (exactly the ones underlying Figure 13, imbalanced) as observed by LP_1 of our example to the normal distribution (a) and to the exponential distribution (b). The parameters for $LSLF(Q^2_{N(0,1)})$ are derived as $b_1(LSLF(Q^2_{N(0,1)})) = 1.26231$ and $b_0(LSLF(Q^2_{N(0,1)})) = 1.89463$; whereas $b_1(LSLF(Q^2_{exp(\lambda)})) =$

0.67198 and $b_0(LSLF(Q^2_{exp(\lambda)})) = 0.63522$. We observe a better LS linear fitting for $Q^2_{N(0,1)}$ since $SSE(Q^2_{N(0,1)}, LSLF(Q^2_{N(0,1)})) = 2.38026 < SSE(Q^2_{exp(\lambda)}, LSLF(Q^2_{exp(\lambda)})) = 4.73942$). Since the $LSLF$ of the normal distribution Q^2 is better than the one imposed by the exponential distribution Q^2 (which can directly be seen from the plots in Figure 14), at least for the prediction based on the particular observation window it is more significant to assume $N(\mu, \sigma^2)$ as the underlying pdf. (Note that for Figure 14 (a) the approximation $x_i = 4.91[q_i^{0.14} - (1-q_i)^{0.14}]$ was used to compute the quantiles for the normal distribution.)

The parameters of the assumed $N(\mu, \sigma^2)$ are determined as follows: For CDFs $F_k(x)$ for which the parameters have the effect of scaling the quantile, it is *not* necessary to explicitly know them; it is sufficient to know only the type of the distribution. For example drawing Q^2 for $N(0,1)$ as in Figure 14 provides the parameters μ and σ^2 of the underlying $N(\mu, \sigma^2)$ distribution as the intercept and the slope of $LSLF(Q^2_{N(0,1)})$; i.e. $\mu = b_0(LSLF(Q^2_{N(0,1)})) = 1.89463$ $\sigma = b_1(LSLF(Q^2_{N(0,1)})) = 1.26231$. For estimating $\widehat{tt}$ we would hence draw an $N(1.89463, 1.26231)$ variate, the confidence value would be $\Phi = 1 - scale(\frac{\sigma}{\mu})$.

The requirements of the prediction method based on Q^2 can be summarized as follows: memory requirements range from $n(N-1)$ if precomputed quantiles are stored, to $2n$ if quantiles are computed from $F^{-1}(x)$ on demand (n for O and another n for the theoretical quantiles in order to construct a regression model). The computational complexity for sorting the observation vector is $\mathcal{O}(n\ ln(n))$ and $\mathcal{O}(n)$ for computing R^2. In order to avoid the storing and resorting of O whenever the observation window moves one step forward, we can use a heuristic [Jain 85] that allows the calculation of quantiles as the observations are appearing (on-the-fly) without storing them. This method reduces the memory requirements to a fixed amount of $2n$.

6 Conclusions

In this work we have developed a *probabilistic* distributed discrete event simulation strategy as a performance efficient compromise between the two classical approaches in parallel and distributed simulation, the *conservative* and the *optimistic* approach. Starting with some theoretical arguing on a sample simulation model we have motivated the potential performance improvement of weakening the conservative "block until safe-to-process"-rule into a "block only if there is no chance of doing useful simulation work"-rule. This is equivalent to impose a local virtual time window on the optimistic strategy, preventing the simulator from propagating incorrect computations too far ahead into the simulated future. The motivation was strengthend by analyzing the distributed simulation performance of the two strategies as observed from execution traces generated on a state-of-the-art parallel processing platform, the Intel iPSC/860. With these traces we could identify, that in order to make DDES practically tractable on contemporary distributed memory multiprocessors, an approach with reduced synchronisation and communication requirements is demanded. The proposed probabilistic synchronisation protocol considerably reduces logical process interactions in exchange for a slightly higher computational complexity, which is neglectible considering the processing power of the target CPUs. Control and data structures of the probabilistic protocol are described. Incremental statistical methods for the characterization of the arrival process for prediction of next external event occurrence times are developed, allowing an 'on-the-fly' forecast at moderate memory and computational costs.

With current work on using autoregressive-integrated moving average processes (ARIMA) for the forecast of next external events based on an observation vector we could improve the precision of predictions, but suffered from high computational complexity. The same experience was made with methods from machine learning. A further paper will report in detail on the tradeoff of attainable forecast precision and its complexity, as well as on the 'appropriate' choice of the arrival history length.

References

[Bell 90] S. Bellenot. "Global virtual time algorithms". In: *Proceedings of the Multiconference on Distributed Simulation.*, pp. 122 – 127, 1990.

[Ben 90] M. Ben-Ari. *Principles of Concurrent and Distributed Programming.* Prentice Hall, Hertfordshire, UK, 1990.

[Chio 91] G. Chiola. "GreatSPN1.5 Software Architecture". In: *Proc. of the 5th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation. Torino, Italy, Feb 13-15, 1991*, pp. 117 –132, 1991.

[Chio 93a] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. "Generalized Stochastic Petri Nets: A Definition at the Net Level and its Implications". *IEEE Transactions on Software Engineering*, Vol. 19, No. 2, Feb. 1993.

[Chio 93b] G. Chiola and A. Ferscha. "Distributed Simulation of Timed Petri Nets: Exploiting the Net Structure to Obtain Efficiency". In: M. Ajmone Marsan, Ed., *Proc. of the 14^{th} Int. Conf. on Application and Theory of Petri Nets 1993, Chicago, June 1993*, pp. 146 – 165, Springer Verlag, Berlin, 1993.

[Chio 93c] G. Chiola and A. Ferscha. "Exploiting Timed Petri Net Properties for Distributed Simulation Partitioning". In: *Proceedings of the 26^{th} Hawaii Int. Conf. on Systems Sciences*, pp. 194 – 203, IEEE Computer Society Press, 1993.

[Fuji 90] R. M. Fujimoto. "Parallel Discrete Event Simulation". *Communications of the ACM*, Vol. 33, No. 10, pp. 30–53, October 1990.

[Gafn 88] A. Gafni. "Rollback Mechanisms for Optimistic Distributed Simulation Systems". In: *Proc. of the SCS Multiconference on Distributed Simulation 19*, pp. 61–67, 1988.

[Jain 85] R. Jain and I. Clamtac. "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations.". *Communications of the ACM*, Vol. 28, No. 10, pp. 1076 – 1085, Oct. 1985.

[Jeff 85a] D. A. Jefferson. "Virtual Time". *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, pp. 404–425, July 1985.

[Jeff 85b] D. R. Jefferson and H. Sowizral. "Fast Concurrent Simulation Using the Time Warp Mechanism.". In: *Proceedings of the Conf. on Distributed Simulation*, pp. 63 – 69, 1985.

[Kona 91] P. Konas and P.-C. Yew. "Parallel Discrete Event Simulation on Shared-Memory Multiprocessors". In: A. H. Rutan, Ed., *Proceedings of the 24^{th} Annual Simulation Symposium, New Orleans, Louisiana, USA, April 1-5, 1991.*, pp. 134–148, IEEE Computer Society Press, 1991.

[Lin 90] Y.-B. Lin and E. Lazowska. "Exploiting Lookahead in Parallel Simulation". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 4, pp. 457 – 469, Oct. 1990.

[Misr 86] J. Misra. "Distributed Discrete-Event Simulation". *ACM Computing Surveys*, Vol. 18, No. 1, pp. 39–65, March 1986.

[Nico 88] D. M. Nicol. "Parallel Discrete-Event Simulation OF FCFS Stochastic Queueing Networks". In: *Proceedings of the ACM/SIGPLAN PPEALS 1988*, pp. 124 – 137, 1988.

[Prak 91] A. Prakash and R. Subramanian. "Filter: An Algorithm for Reducing Cascaded Rollbacks in Optimistic Distributed Simulation". In: A. H. Rutan, Ed., *Proceedings of the 24^{th} Annual Simulation Symposium, New Orleans, Louisiana, USA, April 1-5, 1991.*, pp. 123–132, IEEE Computer Society Press, 1991.

[Sama 85] B. Samadi. *Distributed Simulation algorithms and performance analysis.* PhD thesis, University of California, Los Angeles, 1985.

[Soko 88] L. M. Sokol, D. P. Briscoe, and A. P. Wieland. "MTW: a strategy for scheduling discrete simulation events for concurrent execution.". In: *Proc. of the SCS Multiconf. on Distributed Simulation*, pp. 34 – 42, 1988.