

# Adaptive Time Warp Simulation of Timed Petri Nets

Alois Ferscha, *Member, IEEE*

**Abstract**—Time Warp (TW), although generally accepted as a potentially effective parallel and distributed simulation mechanism for timed Petri nets, can reveal deficiencies in certain model domains. Particularly, the unlimited optimism underlying TW can lead to excessive aggressiveness in memory consumption due to saving state histories, and waste of CPU cycles due to overoptimistically progressing simulations that eventually have to be “rolled back.” Furthermore, in TW simulations executing in distributed memory environments, the communication overhead induced by the rollback mechanism can cause pathological overall simulation performance. In this work, an adaptive optimism control mechanism for TW is developed to overcome these shortcomings. By monitoring and statistically analyzing the arrival processes of synchronization messages, TW simulation progress is probabilistically throttled based on the forecasted timestamp of forthcoming messages. Two classes of arrival process characterizations are studied, reflecting that a natural tradeoff exists among the computational and space complexity, and the respective prediction accuracy: While forecasts based on metrics of central tendency are computationally cheap but yield inadequate predictions for correlated arrivals (thus negatively affecting performance), time series based forecast methods give higher prediction accuracy, but at higher computational cost. The sensitivity of the adaptive optimism control with respect to forecast accuracy and computational overhead is analyzed for very large Petri net simulation models executed with the TW protocol on the Meiko CS-2 multiprocessor, and for a stress case scenario on the CM-5.

Empirical evidence is delivered showing that: 1) probabilistic optimism control, regardless of the communication-computation speed ratio of the target execution platform, automatically finds the most appropriate synchronization policy in the spectrum between optimistic TW and conservative Chandy/Misra/Bryant schemes, 2) local control decisions yield an efficient exploitation of simulation model parallelism that is “local” to particular spatial regions, and 3) even if simulation progresses in “phases” of different performance behavior (nonstationary simulations), logical processes can dynamically readjust their synchronization policy, thus in a natural way evading the partitioning problem under imbalanced loads.

**Index Terms**—Adaptive distributed simulation, Petri nets, Time Warp, optimism control, CS-2, CM-5.

**CR Categories and Subject Descriptors:** 1) C.2 (**Computer Communication Networks:** distributed systems—distributed applications), 2) C.4 (**Computer Systems Organization:** performance of systems—modeling techniques), and 3) I.6.8 (**Simulation and Modeling:** types of simulation—distributed, parallel).



## 1 INTRODUCTION

FOR the quantitative analysis of physical time dynamic systems large timed Petri net models, traditional evaluation techniques like analysis or sequential discrete event simulation tend to become practically intractable. To be able to cope with very large models, parallel and/or distributed simulation mechanisms are demanded and software tools are necessary for the massively concurrent execution of such models.

A variety of approaches has been followed in the literature to adopt standard parallel and distributed simulation techniques [21] to the concurrent execution [13] of Petri nets. *Parallel* simulation strategies exploiting algebraic properties of timed transition firing semantics have been followed by [2], [3], whereas the synchronism of time progression in Petri nets with discrete timing was exploited in [29] to build a massively parallel simulator dedicated to

SIMD execution. *Distributed* simulation approaches involve the spatial decomposition of timed Petri nets and the asynchronous parallel execution of submodels by so called *logical processes* (LPs). LP simulations of Petri nets have been developed along conservative Chandy/Misra/Bryant (CMB), as well as along the optimistic Time Warp (TW) inter LP causality preservation protocols [15]. CMB protocols execute in each LP events that occur in the respective submodel in nondecreasing order of their occurrence timestamp, while strictly preventing from the possibility of any event causality violation across LPs. Such protocols have been adopted for the concurrent execution of Petri nets in [31], [26], [7], [25], [9]. The primary motivation for TW protocols is that events which occur in different LPs—irrespective of their occurrence timestamp—might not affect one another, thus giving reason for their parallel, out-of-order execution. This view has led to TW based distributed Petri net simulators as in [1], [8].

Both CMB and TW protocols have convincing advantages, but also suffer from shortcomings. While CMB protocols rely on the detection of when it is (causally) *safe* to process an event, TW is not reliant on any information coming from the simulation model (e.g., lookahead). Instead, while letting causality errors occur, TW employs a *rollback* mechanism to

• A. Ferscha is with the Institut für Angewandte Informatik, Universität Wien, Lenaugasse 2/8, A-1080 Vienna, Austria.  
E-mail: ferscha@ani.univie.ac.at.

Manuscript received 1 Oct. 1997; revised 12 May 1998.

Recommended for acceptance by G. Chiola and W.H. Sanders.

For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 107270.

recover from causality violations immediately upon detection (*aggressive cancellation*), or after local resimulation (*lazy cancellation*). The rollback procedure in turn relies on the ability to reconstruct past states, which can be guaranteed by a systematic state saving policy and corresponding state reconstruction procedures. Memory management has a crucial performance impact in TW implementations [24]. Particularly, the possibility of releasing storage occupied for simulation objects (state variables, synchronization messages, etc.) that have been *committed* not to become subject of rollback in the future is important to prevent from simulation stalls due to memory depletion. To this end, an estimate of the global virtual time GVT (at least a lower bound) is demanded in each LP, such that a mechanism called *fossil collection* can relocate memory from the simulation history log that will no longer be used. Estimating GVT itself is a nontrivial and time consuming TW overhead [15].

The TW specific state saving overheads are not present in CMB protocols, but the strict adherence to time-stamp-order event processing makes these protocols prone to deadlock due to cyclic waiting for safe-to-process concessions. A deadlock management mechanism is necessary to either avoid deadlocks, or detect and recover from deadlocks. In distributed implementations of CMB protocols deadlock management is usually based on the exchange of messages (to make 'unsafe' events safe to process by exploiting information from their timestamps), yielding severe communication overheads. Deadlock management is relieved from TW in a natural way, since deadlocks due to cyclic waiting conditions can never occur. Nevertheless can TW suffer from communication overheads to a threatening extent: In situations where event occurrences are highly dispersed in space and time, rollback invocations can recursively involve long cascades of LPs which will eventually terminate. An excessive amount of local and remote state restoration computations is the consequence of the annihilation of effects that have been diffused widely in space and too far ahead in simulated time, consuming considerable amounts of computational, memory and communication resources while not contributing to the simulation as such. This pathological behavior is basically due to the "unlimited" optimism assumption underlying TW, and has often been referred to as *rollback thrashing*. Overall, no general rule of superiority of the two strategies can be formulated [15], and the analysis of the coinfluence of factors determining performance is—due to the overwhelming complexity of factor interweaving—an open research issue [20].

Several approaches for finding a compromise between CMB and TW have been followed in the parallel and distributed simulation literature. In [27] the possibilities of options for combining CMB and TW to attain the capabilities of both are classified as:

- 1) *relaxing conservatism* in CMB protocols,
- 2) *limiting the optimism* in TW protocols, and
- 3) *seamlessly switching* between optimistic and conservative schemes, establishing the class of *hybrid protocols* [10].

In the domain of parallel and distributed simulation of Petri nets, the issue of CMB vs. TW is also not conclusive, which

has lead us to raise the question for a hybrid protocol approach in [7]. Mainly two reasons have motivated the development of so called "*adaptive*" protocols [14], i.e., hybrid protocols that can adjust to any point in the continuum between conservative and optimistic extremes, CMB and TW:

- 1) the identification and appropriate setting of performance control parameters is prohibitively difficult for a simulation modeler who is not intimately familiar with the raw performance of the target architecture and the distributed simulation protocol implementation intrinsics, and
- 2) many simulation models exhibit time varying runtime characteristics, necessitating mechanisms of automated control parameter readjustment.

An instance of the latter, namely that a static (spatial) partitioning of the Petri net simulation model is insufficient for performance optimization, will be outlined in this work.

**Outline.** This paper, after explaining *logical process simulations* of timed Petri nets conceptually, and presenting related work on optimism control in TW (Section 2), recalls the adaptive TW protocol developed for the efficient concurrent execution of large Petri net simulation models. The key idea behind the adaptive protocol is that control (throttling) decisions are made "on-the-fly," based on the exploitation of implicit model parallelism derived from a statistical message arrival process analysis and message timestamp prediction (we shall refer to the amount of potentially simultaneous occurrences of transition firings as model parallelism, and intend to execute as many as possible of those in different logical processes). In Section 3, a collection of such analysis methods is developed at different levels of sophistication and with increasing computational complexity. In a case study in Section 4, the performance sensitivity of these forecast methods is evaluated with a very large Petri net business process model. To illustrate the sensitivity of the simulator performance with respect to the various forecast methods also a stress case example is constructed and analyzed. Conclusions are drawn in Section 5.

## 2 ADAPTIVE LOGICAL PROCESS SIMULATION

### 2.1 Logical Process Simulation of Petri nets

A *logical process simulation* (or *distributed simulation*) of timed Petri nets (TPNs) employs a set of LPs to execute event occurrences of a spatially partitioned TPN graph asynchronously in parallel on different nodes of a multiprocessor system. The software architecture of a logical process LP<sub>*i*</sub> comprises

- 1) an event list (EVL) based *simulation engine* SE<sub>*i*</sub>, where transitions  $t_i \in T$  are scheduled for firing at their occurrence time  $ot(t_i)$ ,
- 2) a *communication interface* CI<sub>*i*</sub> connecting LP<sub>*i*</sub> to other LPs and providing the possibility to synchronize local events with remote events, and
- 3) an assigned *region* R<sub>*i*</sub> as (spatial) part of the TPN graph. SE<sub>*i*</sub> operates on R<sub>*i*</sub> in an event driven mode, i.e., it executes *local* events (transition firings), and generates *local* and *remote* future events (transition firings).

With the execution of events,  $SE_i$  progresses a *local clock* usually referred to as local virtual time (LVT), i.e., the simulated time in the respective region  $R_i$ . At a certain point of (wall clock) time, the various LPs, therefore, may have progressed to different instants of LVT, thus defining an asynchronous distributed execution and exposing a synchronization problem for the scheduling of events in remote LPs. Each LP $_i$  ( $SE_i$ ) has access only to a statically partitioned *subset of the state variables*  $S_i \subset S$ , constituted by the token counts in places  $p \in P$  contained in  $R_i$  ( $S_i$  is disjoint to state variables assigned to other LPs). Two kinds of events processed in each LP $_i$  are distinguished: *internal events* have causal impact only to  $S_i \subset S$ , whereas *external events* also affect  $S_j \subset S$  ( $i \neq j$ ), the local states of other LPs. The communication interface  $CI_i$  takes care for the propagation of effects causal to events to be simulated by remote LPs, and the proper inclusion of external causal effects. This is done by “encoding” the token(s) released by a transition upon firing for a place (or places) in a remote LP (or LPs) into a token message  $m$  (piggybacked with a copy of the senders LVT at the sending instant), the delivery of  $m$  to the destination LP(s), and the “decoding” of  $m$  into the enabling of transitions and their scheduling in the receivers EVL according to the message timestamp value  $t(m)$ , relative to the receivers LVT.

A crucial performance aspect of TPN logical process simulations under both CMB and TW is the choice of the decomposition of the TPN model in regions. Traditionally [8], the TPN structure is analyzed to exploit simulation model parallelism, which is done in a two step heuristic:

- 1) a decomposition is chosen that prevents from *conflicts* among transitions in different LPs, and
- 2) the resulting “minimum grain size” regions are *packed* to larger grains. Step 1) mainly involves the computation of the extended conflict set  $ECS(t)$  for each  $t \in T$ , i.e., the transitive closure of  $t$  with respect to the symmetric structural conflict relation SCC.

Denote the (initial) marking of a TPN by  $(\mu^{(0)}) \mu$ , the set of transitions enabled in  $\mu$  by  $E(\mu)$ , and the input set of  $t$  by  $I(t)$ , then SCC is developed as follows:

- $t_i, t_j \in T$  are in *structural conflict* ( $t_i SC t_j$ ), iff  $I(t_i) \cap I(t_j) \neq \emptyset$ .
- $t_i, t_j \in T$  are *mutually exclusive*, ( $t_i ME t_j$ ), iff  $\exists \mu$  s.t.  $t_i, t_j \in E(\mu)$ .
- $t_i, t_j \in T$  are in symmetric structural conflict ( $t_i SSC t_j$ ), iff  $((t_i SC t_j) \wedge (t_j SC t_i)) \vee (t_i \neg ME t_j)$ .

The symmetry, transitivity and reflexivity of  $SSC$  now allows to compute  $ECS(t)$  as the equivalence classes of  $t \in T$  with respect to  $SSC$ . A “minimum” spatial region  $R_k$  is then constructed containing a subset of transitions  $T_k \subseteq T$  where

- $T_k = \{t_j\}$ ,  $t_i \in T$  is a single transition, iff  $t_i \in T$  does not participate in any  $ECS$ ,
- $T_k = ECS(t_i) \subseteq T$ ,  $t_i \in T$  is the whole  $ECS$ , iff  $t_i \in T$  participates in some  $ECS$ ,

together with the respective input set  $I(T_k)$ . For packing minimum regions into larger grains the following relations are used:

- $t_i, t_j \in T$  are *causally connected* ( $t_i CC t_j$ ), iff  $t_i \in E(\mu)$  and  $t_j \in E(\mu)$ ,  $\mu \xrightarrow{t_i} \mu'$  might cause that  $t_j \in E(\mu')$ .
- $t_i, t_j \in T$  are *concurrent* ( $t_i CN t_j$ ), if they are neither SC, nor CC, nor ME.

Grain packing heuristics will certainly utilize these structural properties to maximize the amount of potentially simultaneous transition firings among the involved LPs; e.g., given  $(t_i CN t_j)$ , then  $t_i, t_j$  should be in different regions since they do exhibit potential model parallelism, whereas given  $(t_i ME t_j)$ , then  $t_i, t_j$  should be in the same region (no parallelism). Besides the simulation model structure, grain packing needs to take into account target hardware (CPU performance, available memory, caching, etc.) and software (multithreading overhead, scheduling, etc.) characteristics, as well as the communication-computation speed ratio exhibited by the communication system.

Up until now, the grain packing problem has not found efficient solutions. More than that, performance deficiencies due to the dynamics of many simulation models cannot be overcome by static partitioning heuristics based on the considerations above. Consider as an example the trivial stochastic TPN model of a machine failure/repair system in Fig. 1. Two LPs are used to simulate failures (LP $_1$ ) and repairs (LP $_2$ ), both occurring at an exponentially distributed rate, simulated on two dedicated nodes of a CM-5 multiprocessor using the TW protocol. The variation of model parallelism (tokens representing machines), and the imbalance in the model timing (repairs that take twice as long, four times as long etc. than failures) reveals the performance behavior given in Fig. 2.

It is clearly seen, that the different LVT increments in the communicating LPs forces a shift of workload from LP $_1$  to LP $_2$  with increasing degree of imbalance (LP $_1$  with small LVT increments frequently forces LP $_2$  (with high LVT increments) to roll back): the higher the expected enabling time of  $t_2$ , the more tokens will reside in  $p_2$  (in steady state) enabling  $t_2$ . Due to the drain off of tokens from LP $_1$ , simulation operations (EVL management) become “cheaper” in LP $_1$ , making the LP “faster” (percentage of CPU time spent for simulation work decreases, while idle time, i.e., time spent waiting for messages increases). On the other hand, due the token load in LP $_2$ , simulation operations tend to use more and more CPU time, while the time waiting for messages declines. Note that due the synchronization that the TW protocol exposes to the tokens circulating among the two LPs, both LP $_1$  and LP $_2$  conduct exactly the same amount of (committed) simulation work. LP $_2$ , however, uses increasingly more CPU resources for getting the work done.

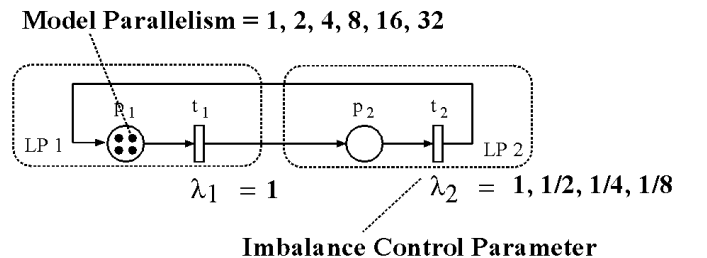


Fig. 1. Machine failure/repair scenario.

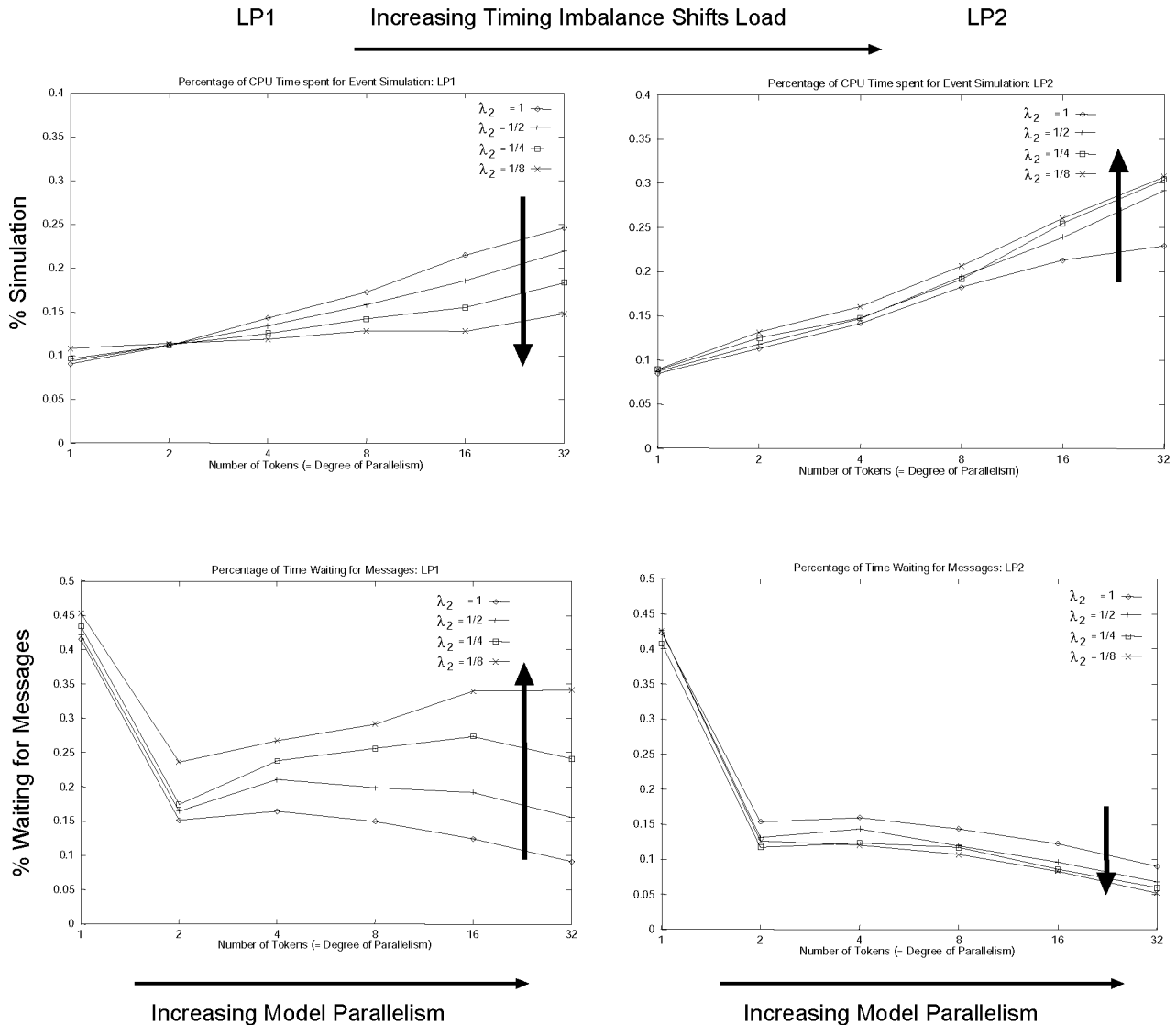


Fig. 2. Dynamic load shift caused by imbalanced model timing (TW on CM-5).

A consequence of this example is, that static partitioning (not even repartitioning at runtime [25]) can suffice to cope with such execution dynamics in a satisfying way. In order to improve overall TW performance, an adaptive mechanism is demanded to throttle the “fast” LP ( $LP_1$ ) to a LVT progression speed that best fits to the progress of the “slower” LP ( $LP_2$ ), or in other words, the CPU cost for simulation operations needs to be balanced among the involved LPs. The method of choice for this purpose (in this paper) is a probabilistic, adaptive optimism control for TW, based on implicit throttling.

## 2.2 Adaptive Optimism Control in Time Warp

The need for adaptivity of distributed simulation protocols has early been seen in the literature [28], and was recently summarized [10]. In the context of TW, the Adaptive Time Warp (ATW) protocol [4] allows the execution of events  $e$  with timestamp  $t(e)$  only if  $t(e) \in [t, t + \Delta)$ , where  $\Delta$  is adapted dynamically according to the number of local causality constraint violations (rollbacks) in the past. In [17] a protocol is proposed that temporarily blocks an LP if a po-

tential straggler messages (that would induce rollback) is anticipated. The Local Adaptive Protocol (LAP) [22] in much the same way uses statistical information from observing the simulation performance on-the-fly to dynamically adjust optimism control parameters. An explicit cost function to determine whether it is more “cost effective” to execute an arriving message instantaneously despite rollback hazard, or to delay its execution to avoid potential rollback/antimessages is reported in [18]. A combination of controlling optimism, and an automatic adjustment of the amount of memory required in TW is approached in the Adaptive Memory Management (AMM) scheme [11]. All these protocols use statistical data that reflects the central tendency (e.g., average increase of local virtual time and average (real and simulated) message interarrival time in LAP; average commitment rate in AMM) to determine parameters for optimism control in TW. However, optimism control based on averages is not promising when simulation models incur “phases” of different LP synchronization behavior, where reactivity and an automatic readjustment of dynamically evolving changes in the simulation workload control parameters from “phase” to “phase” is desired.

In [14], we have developed an LP synchronization protocol which combines the potentials of CMB and TW in a probabilistic way. The adaptive scheme, as opposed to CMB which would block the CPU if there is the chance of a message with a timestamp in the past (straggler message) arriving at the LP, and opposed to TW which would optimistically progress LVT even if the chance for receiving a straggler message is very high, executes scheduled events based on the probability of not receiving a straggler. Assume (Fig. 3) the timestamp of the *next* (token-)message  $m_{i+1}$  arriving at some LP to be equally likely in an (arbitrary wide) interval  $[s, t]$ . In this case, the LP can safely simulate scheduled firings  $\langle t_k @ ot(t_k) \rangle$  with  $ot(t_k) < t(m_{i+1})$ . The CMB protocol blocks at  $LVT = s$ , and by that fails to use a  $(1 - \epsilon)$  chance by not executing events scheduled in the virtual time period  $[s, \epsilon(t - s)]$ . This must be considered as an *overpessimism* at least for small  $\epsilon$ . TW on the other hand executes even firings with  $ot(t_k) > t$  which will definitely (with probability 1) have to be rolled back. This is a clear *overoptimism*, and the rollback(s) and communication overhead for sending annihilation messages for any  $t_k$  with  $ot(t_k) > t$  could clearly have been avoided. The question arises, whether it is more “cost effective” to execute scheduled events instantaneously despite the increasing rollback hazard, or delay the execution to avoid potential rollback/antimessages.

The probabilistic adaptive scheme reduces the waste of CPU cycles due to blocking in CMB, while at the same time reducing the communication cost caused by annihilation messages in case of rollback in TW. Technically, the degree of “optimism” underlying TW is regulated according to hypotheses that LPs are establishing during simulation on the amount of available model parallelism as observed from the timestamps carried by arriving messages ( $t(m_{n+1}), t(m_n), \dots t(m_i)$ ). Given the timestamp of the forthcoming message  $t(m_{i+1})$  would be

known a priori, then all the transitions  $t_k$  scheduled in EVL with  $LVT \geq ot(t_k) < t(m_{i+1})$  could be safely executed (we have referred to the number of transitions  $t_k$  with  $LVT \geq ot(t_k) < t(m_{i+1})$  as “implicit model parallelism”). Practically therefore, with the help of an estimate  $\hat{t}(m_{i+1})$  for the forthcoming message, the “aggressiveness” of TW can be adaptively throttled to that point in the spectrum between unlimited optimism and extreme pessimism, that is the best compromise for a particular simulation model. Since  $\hat{t}(m_{i+1})$  is just an estimate, we have to relate the control decision to the confidence in  $\hat{t}(m_{i+1})$ : assuming the confidence in the forecast to be  $0 < (1 - \alpha) < 1$ , then a transition  $t_k$  scheduled with  $ot(t_k)$  is processed with probability

$$P[\text{process } t_k] = 1 - \frac{1}{1 + e^{-\frac{LVT_j - \hat{t}(m_{i+1})}{\alpha(1-\alpha)100}}}$$

otherwise its execution is delayed (the CPU is blocked) for the average amount of CPU time required to simulate one event. Fig. 3 explains the blocking probability related to the confidence level  $(1 - \alpha)$ : The higher the confidence, the steeper the ascent of the delay probability as LVT progresses towards  $\hat{t}$ . (Steepness of the sigmoid function in Fig. 3 with  $(1 - \alpha) = 0.95$  is higher than with  $(1 - \alpha) = 0.90$ ). After LVT has surpassed  $\hat{t}$ , delays become more and more probable, expressing the increasing rollback hazard the LP runs into. A general observation is that with  $(1 - \alpha) \approx 1$ , throttling yields a synchronization behavior close to CMB, whereas with  $(1 - \alpha) \approx 0$ , optimism is as unlimited as in TW.

### 2.3 The Adaptive Time Warp Simulation Engine

A sketch of the TW protocol is given in Fig. 4, with conditional compile directives for the adaptive (throttled) version. During initialization (s1) the assigned region  $R_i$  of the

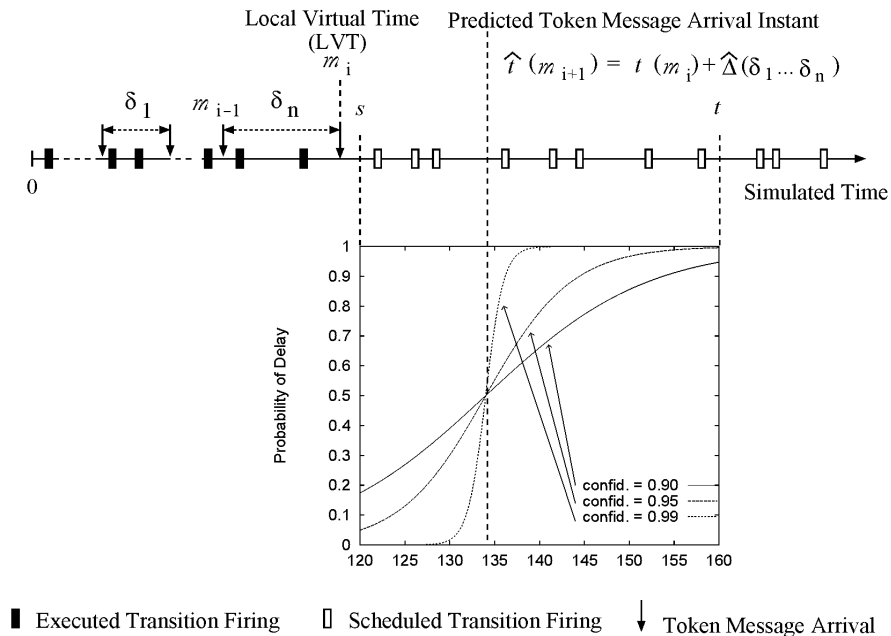


Fig. 3. Adaptive optimism throttling based on message timestamp prediction.

```

Simulation_Engine(i) {
s1  iel=initial(Ri);
    while(ie=next_ie(iel)) chronological_insert(ie,EVL);
    log_new_state();
s2  while (GVT < ENDTIME {
s3    while (m=read_next_input_buffer_message()) {
        #if THROTTLED
s4      if (positive(m)) update_statistics(m);
        #endif
s5      if ( t(m) < LVT) process_Straggler(m);
s6      if (!remove_dual(m,IQ)) insert(m,IQ);
    }
    #if THROTTLED
s7      estimate(LVTH, confidence);
s8      if (get_time_first_EVL_or_IQ() ≥ LVTH)
s9        if (random() < confidence)
            delay(AVE_EVENT_PROC_TIME); continue;
        #endif
s10     advance_GVT(); fossil_collection();
s11     if (memory_used > MEMORY_LIMIT) continue;
s12     e = get_first_EVL_or_IQ();
        if (e) process_event(e);
s13     fill_OB(OQ);
        send_out_contents(OB);
    }
s14  analyse_stack(); clean_up();
}

```

Fig. 4. Throttled Time Warp simulation engine.

Petri net description file is read, and model initialization produces a list of initial internal events, which are inserted into EVL in chronological order. The resulting initial state is subsequently logged on the simulation state stack. The main simulation loop is then executed until GVT reaches the value ENDTIME (s2). Incoming messages are read from the LP's input buffer (s3), and their timestamps are checked against the current LVT (s5). If the timestamp of *m* is smaller than LVT, maybe a rollback has to be induced. To verify a causality violation, the *input queue* IQ is searched for dual messages ((r1) of **process\_Straggler(m)** in Fig. 5). (The dual of a positive message is an antimessage and vice versa.) A rollback needs to be performed if *m* is a positive message and no corresponding antimessage has been received before (this can happen if message delivery is not FIFO), or if *m* is an antimessage and its corresponding positive message has been received before (a dual message exists)(r2). Only in those cases the rollback mechanism needs to be invoked to restore the first consistent state prior to the timestamp of the straggler message (r3). Antimessages are generated in the course of rollback (r4), and inserted into the *output queue* OQ (r4) (the lazy cancellation protocol does not send them immediately). As a final action

of the input processing part, each incoming message is inserted into the input queue IQ, or annihilated if it finds its dual counterpart in IQ (s6).

In step (s10) the algorithm tries to progress GVT, and if successful, cleans up fossils from the simulation history log. GVT is calculated in a distributed way, involving the sending of packets containing a vector of GVT estimates from all LPs: If a GVT calculation packet has been received in LP<sub>*i*</sub> during the input phase, then the **advance\_GVT()** code segment is executed. Using the information contained in the packet, LP<sub>*i*</sub> calculates a new GVT estimate, updates its own information in the GVT packet and forwards it to its successor.

Step (s11) checks if sufficient memory (specified in MEMORY\_LIMIT) is available to perform the local simulation of the next event. If not, the simulator loops back to the input section (s2) to await the arrival of further messages or GVT calculation packets until sufficient memory is freed through the occurrence of a rollback (arrival of a straggler message) or fossil collection (arrival of a GVT calculation packet). In the case that no events are scheduled for local simulation in the IQ or EVL, the simulator also loops back to the input section (s3). The occurrence of the next scheduled event is simulated by modifying the LPs state variables accordingly ((e1) of **process\_event(e)** in Fig. 5). **modified\_by\_e(e)** in (e1) returns three lists of events:

- 1) a list of the new internal events resulting from the occurrence of the event in the model which are to be scheduled for future simulation,
- 2) a list of previously scheduled events which have now been preempted by the occurrence of the event and
- 3) a list of new external events (message arrivals in other partitions) which must be sent to the respective processors.

The state of the simulator is updated to reflect the occurrence of the event by inserting the internal events into the EVL (e2) and removing the preempted events (e3). External events are inserted into the output queue (OQ) or alternately annihilated if a dual message is present in the OQ (lazy cancellation) (e5). In (e4) the current state of the execution is saved by copying the EVL to the state stack as well as the state variables used by the model. Finally, messages stored in the OQ with timestamps less or equal to the current LVT are moved to the output buffer and sent to the respective LPs in (s13) and control loops back to the input phase. The simulation run terminates when GVT reaches ENDTIME. Upon loop termination, the state stack is analysed and performance data is gathered (s14).

The "throttled" version of the protocol uses all the code of the "plain" TW implementation, and thus precludes a source of bias in the performance comparison of the two. In (s4) the throttled protocol collects message arrival statistics, which are used in the forecast procedure in (s7). **estimate()** returns a local virtual time horizon (LVTH) defined by the estimated timestamp of the forthcoming message, and the level of confidence in that forecast. The SE in (s8) and (s9) probabilistically decides to delay the processing of internal events (transition firings) or arriving messages. After blocking, control immediately loops back to the SE input section to check for new arrivals of messages, which could

```

process_Straggler(m) {
r1  dual=dual_exists(m, IQ);
r2  if ((positive(m)&& !dual) || (negative(m)&& dual))
    {
r3   LVT=restore_earliest_state_before(t(m));
r4   ant_evl=generate_antimessages(LVT);
      while(ee=next_ee(ant_evl))
        chronological_insert(ee, OQ);
    }
}

process_event(e) {
e1  new_ev, pre_evl, ext_evl =modified_by_e(e);
      while(ie=next_internal_e(new_evl))
e2   chronological_insert(ie, EVL);
      while(ie=next_preempted_int_e(pre_evl))
e3   remove_event(ie, EVL);
e4  LVT = t(e);log_new_state();
      while(ee=next_external_e(ext_evl))
e5   if (!dual_update(ee, OQ))
        chronological_insert(ee, OQ);
}
    
```

Fig. 5. Rollback procedure and event processing.

be stragglers that deserve highest processing priority. (A few variations of this policy will be investigate in the experiments presented below.)

### 3 PREDICTING TIMESTAMPS OF FORTHCOMING TOKEN MESSAGES

A critical issue with the adaptive protocol is the way in which  $\hat{t}(m_{i+1})$  is computed: Forecasts based on the central moment of the arrival history will cause less computational overhead, but may generate inadequate predictions, whereas methods based on a time series analysis with the potential of giving more adequate predictions for particular arrival processes can become excessive in memory and CPU cycle consumption. Through the rest of this paper we shall investigate a variety of methods for computing  $\hat{t}(m_{i+1}) = t(m_i) + \hat{\Delta}(\delta_1, \delta_2, \dots, \delta_n)$  from the observed token timestamp differences  $\delta_k = t(m_{i-n+k}) - t(m_{i-n+k-1})$ .

It is obvious that the overall performance of adaptive throttling in TW with the approach presented above is reliant on the quality of the predictor  $\hat{t}$ , and the confidence  $\alpha = \zeta(\hat{t})$  in the predictor. This quality is in turn influenced by the amount of information available for forecasting, i.e., the size of the observation window  $n$  maintained for each input channel in every LP, as well as the choice of the forecast procedure. Generally, the larger  $n$ , the more information on the arrival history is available in the statistical

sense. Considering much of the arrival history will at least theoretically give a higher prediction precision but will also consume more memory space. Intuitively, complex forecast methods could give “better” predictions than trivial ones, but are liable to intrude on the simulation engine with an unacceptable amount of computational resource consumption. Therefore, *incremental* forecast methods of low memory complexity are recommended, i.e., procedures where  $\hat{t}(m_{i+2})$  can be computed from the previous forecast  $\hat{t}(m_{i+1})$  and the current observation  $t(m_{i+1})$  in  $O(c)$  instead of  $O(cn)$  time.

#### 3.1 Central Tendency Forecasts

##### 3.1.1 Arithmetic Mean Based Forecasting

A straightforward estimation method for  $\hat{t}(m_{i+1})$  is arithmetic averaging, i.e., to use the arithmetic mean of time increments  $\hat{\Delta} = \bar{\delta}$ , with  $\zeta(\hat{t}) = 1 - \frac{s}{\bar{\delta}}$ , where  $s$  is the empirical standard deviation of  $\delta_1, \delta_2, \dots, \delta_n$ . Both  $\bar{\delta}$  and  $s$  can be computed incrementally, i.e., given  $\bar{\delta}_n$  and  $s_n$  after  $n$  messages. Upon a new arrival carrying a time increment  $\delta_n$ , the new  $\bar{\delta}_{n+1}$  and  $s_{n+1}$  are computed as

$$\bar{\delta}_{n+1} = (n\bar{\delta}_n + \delta_{n+1}) / (n + 1). \quad (1)$$

$$s_n^2 \stackrel{\text{def}}{=} \frac{1}{n-1} \sum_{i=1}^n (\delta_i - \bar{\delta}_n)^2 = \frac{n(\bar{\delta}_n^2 - \bar{\delta}_n^2)}{n-1} \quad (2)$$

$$\Rightarrow s_{n+1}^2 = \frac{n+1}{n} (\bar{\delta}_{n+1}^2 - \bar{\delta}_n^2) \quad (3)$$

The main advantage of using the arithmetic mean as a forecast is its incremental computation involving  $O(1)$  operations. (Note that for the incremental computation of  $s^2$  also  $\bar{\delta}^2$  needs to be computed incrementally.) A potential disadvantage appears when the distribution of  $\delta_i$  is skewed—the prediction would then either consistently overestimate or underestimate the next token time. In this case, but also if the sequence contains randomly occurring asymmetric outliers, a forecast based on the median appears more appropriate.

##### 3.1.2 Median Based Forecasting

Median computation over a sliding window of size  $n$  involves sorting the vector  $X$  of observed message time increments. The forthcoming message time is predicted to be  $\hat{t} = t_n + \hat{\Delta}$ , where  $\hat{\Delta}$  is the median of the time increments. Let  $X$  contain the sorted vector  $(\delta_1, \delta_2, \dots, \delta_n)$ , the median (or 50 percent quantile  $q_{.50}$ ) is defined as:

$$Me(X) = \begin{cases} X_{\lfloor \frac{n+1}{2} \rfloor}, & \text{if } n = \text{odd} \\ \frac{1}{2} \left( X_{\lfloor \frac{n}{2} \rfloor} + X_{\lfloor \frac{n}{2} \rfloor + 1} \right), & \text{if } n = \text{even} \end{cases} \quad (4)$$

The primary advantage here is the median property of the smallest possible deviation:

$$\sum_{i=1}^n |X[i] - Me(X)| \leq \sum_{i=1}^n |X[i] - c| \quad \forall c \in \mathbb{R},$$

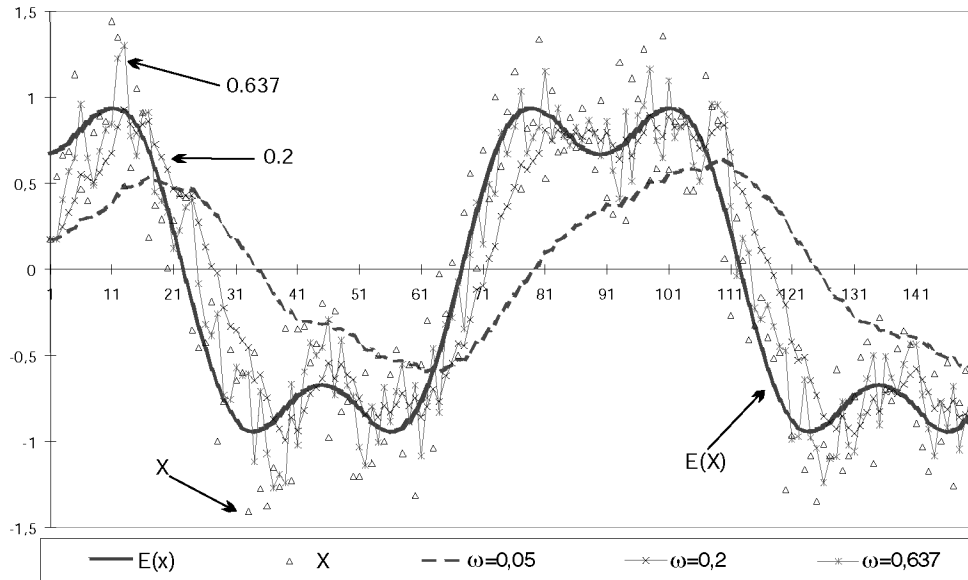


Fig. 6. Effect of exponential smoothing.

and the robustness against outliers in the arrival stream, i.e., it rejects the effects of sharply outlying timestamps from the forecast they produce. Instead of the standard deviation the *interquartile range* ( $q_{.75} - q_{.25}$ ) is used as the forecast confidence information. The disadvantage of median based forecasts is that it prohibits an incremental computation, causing an  $O(\log(n))$  sorting effort in each step. (An incremental median approximation alleviating this shortcoming is studied in [14] attempting to express the relation between mean and median as  $\alpha = \frac{Me(\delta)}{\mu(\delta)}$ , where  $\alpha$  and the median are computed only after every a fixed number of incoming message. The forecast  $\hat{t} = t_n + \hat{\Delta}$  in this cases uses  $\hat{\Delta} = \alpha * \bar{\delta}$ .)

### 3.1.3 Exponential Smoothing

*Exponential smoothing* is a method to remove seasonal trends from time series and allows for a weighting of the more recent history over the less recent history or vice versa. Here,  $\hat{\Delta}$  is a weighted moving average of  $\delta_k, \delta_{k-1}, \dots$ , with weights decreasing exponentially on the weighting factor  $\omega \in (0, 1)$ , incrementally computed as:

$$\hat{\Delta}_{k+1} = \omega \delta_k + (1 - \omega) \hat{\Delta}_k. \quad (5)$$

To approximate the arrival process as “smooth” as possible,  $\omega$  needs to be chosen so as to minimize the residuals:

$$\min \sum |\delta_k - \Delta_k(\omega)| \quad 0 \leq \omega \leq 1.$$

Practically this minimum is approximated by computing the sum of residuals for  $q * 10$  different values of  $\omega$  within the interval  $[0, 1]$  after every  $n$ th incoming message.

Fig. 6 shows a sample arrival process  $X$  and the effect of varying smoothing parameters  $\omega = 0.05$ ,  $\omega = 0.2$ , and  $\omega = 0.637$ . In the particular example, the parameter  $\omega = 0.637$  minimizes the residuals exposed by  $X$ .

## 3.2 Time Series Forecasting

### 3.2.1 Forecasting Based on ARMA $[p, q]$ Models

In cases where the arrival process  $(\delta_1, \delta_2, \dots, \delta_n)$  exhibits trends and/or seasonal behavior, the previous forecast methods are prone to pathological behavior due to repeated over- and underestimation of the forthcoming message timestamp. To deliver more robust forecasts for arrivals that exhibit phases, the identification of the time series underlying the arrival process is crucial. As an illustration consider the timestamp trace collected from a TW execution of the machine failure/repair model from Fig. 1 on the CM-5 (Fig. 7a) gives the sequence  $X = (\delta_1, \delta_2 \dots \delta_{300})$  for a four machine model with exponential failures (enabling time of  $t_1$  is  $\tau(t_1) \sim \exp(1.0)$ ) and deterministic repair delays (enabling time of  $t_2$  is  $\tau(t_2) = 8.0$ ) as observed by LP<sub>2</sub>). The shape of the corresponding autocorrelation (ACF, Fig. 7b) and partial autocorrelation function (PACF, Fig. 7c) leads us to hypothesize that the arrival process is *autoregressive*, since we find ACF dying down in an oscillating damped exponential way. This is also intuitive because the deterministic component in the process ( $\tau(t_2) = 8.0$ ) dominates the stochastic one ( $\tau(t_1) \sim \exp(1.0)$ ).

For arrival processes which appear as an arbitrary combination of an unknown stochastic process with autoregressive and/or moving average components, we have developed a characterization model based on (*stationary*) ARMA $[p, q]$  processes. ARMA $[p, q]$  is a combination of a pure autoregressive process of order  $p$  (AR $[p]$ ) explaining a time series  $Y_t$  as a dependency  $Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \epsilon_t$ ,  $\epsilon_t$  being a white noise random error, and a pure moving average process of order  $q$  (MA $[q]$ ) that explains  $Y_t$  as a series of i.i.d. white noise errors  $Y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q}$  with  $E(\epsilon_i) = 0$ ,  $\text{Var}(\epsilon_i) = \sigma_\epsilon^2$  and  $E(Y_t) = 0$ . The classical Box Jenkins method [5] containing procedures for

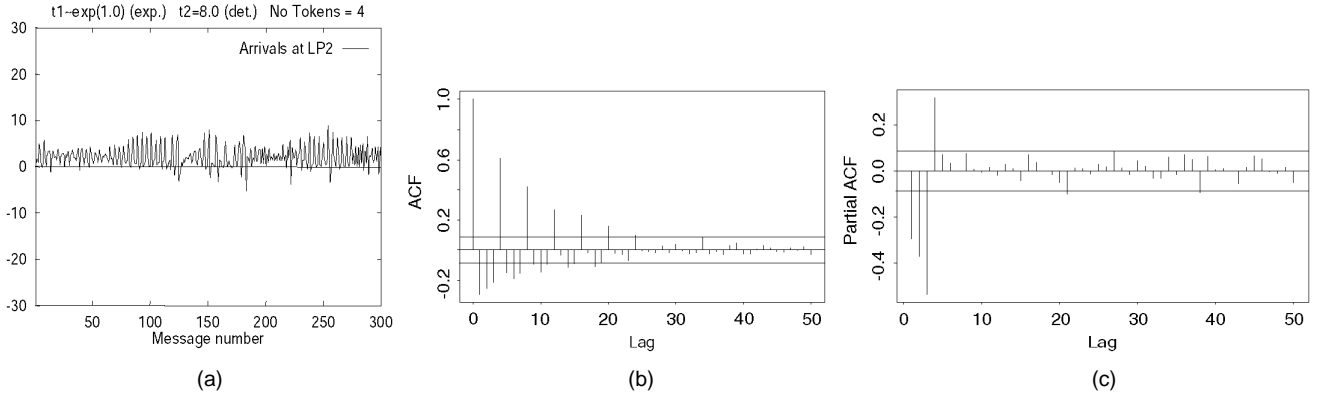


Fig. 7. Message timestamp correlation analysis for the machine failure/ repair model under TW on the CM-5.

- 1) model order identification (determination of  $p$  and  $q$ ),
- 2) model parameter estimation (maximum likelihood estimation of  $\phi_i$  and  $\theta_j$ ),
- 3) model diagnostic and verification (lack-of-fit testing of the residuals), and
- 4) forecasting (Durbin-Levinson recursion [6] has been implemented, see the technical details in Appendix A).

For the particular example series in Fig. 7, the procedure finds the best fitting model to be ARMA[3, 0], based on which the integrated Durbin-Levinson  $k$ -step best linear forecast procedure generates the forecasted continuation of the series as depicted (dashed) in Fig. 8. For the particular example process it is clearly seen that the ARMA model is more adequate than central tendency forecasts, which would suffer from continuous over and underestimation in this correlated timestamp regime.

### 3.2.2 Kalman Filter Forecasting

Yet another approach for estimating the timestamps of forthcoming messages is to use a *Kalman filter model*, which (as an improvement) supports a recursive forecast procedure (technical details are given in Appendix B). Still, both the forecast methods based on time series analysis and the Kalman filter approach suffer from complex computation, which can potentially overwhelm the gain of adaptive rollback prevention. On the other hand can both ap-

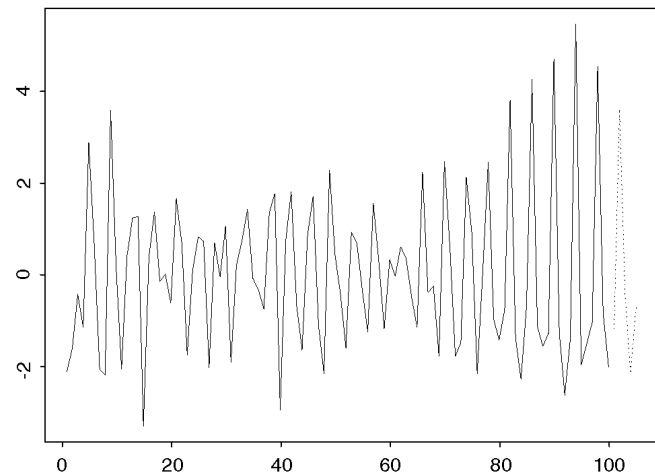


Fig. 8. Durbin-Levinson best linear forecast for the machine failure/repair model series (TW on the CM-5).

proaches provably recognize correlated arrivals and achieve more adequate predictions than methods based on central moments. In the following case studies we shall evaluate the potentials of all of these methods as applied to the distributed TW simulation of very large timed Petri net models, as well as in stress scenarios.

## 4 CASE STUDIES

### 4.1 A Very Large Distributed Simulation Experiment

To investigate the performance sensitivity of adaptive methods, timed Petri net models describing the document flow in a very large business organization were chosen (more than  $10^6$  documents “flowing” through more than 60 offices). To avoid presenting the full complexity of the models in this paper, we just present a simplified abstraction as shown in Fig. 9. In this abstraction, the documents flowing into a work area (office) are represented by tokens arriving at a place  $p_{in}$ . A transition  $t_{ext}$  propagates documents to other offices (regions), or routes them to the appropriate places of execution within that work area, i.e., region. In the experiments we study an office system consisting of 64 offices, connected to each other by the arcs originating from respective  $t_{ext}$ s, and discharging into remote  $p_{in}$  places; i.e., the execution of  $t_{ext}$  in  $LP_i$  causes a document (token) to be transferred to the successor region  $LP_{i+1}$ . If  $LP_i$  and  $LP_{i+1}$  are assigned to different processors, the TW protocol generates and sends a message for each document flowing from  $LP_i$  and  $LP_{i+1}$ , whereas documents flowing within one region are local and do not invoke any message sending (a more detailed description of the kind of models investigated is given in [19], [16]). To introduce sufficient variation with respect to model behaviors, we examine transitions  $t$  adhering to the race firing policy and three different kinds of enabling time  $\tau$

- 1) exponential ( $\tau(t_j) \sim \exp(\lambda_j)$ ),
- 2) deterministic ( $\tau(t_j) = (1/\lambda_j)$ ), and
- 3) mixed ( $\tau(t_j) \sim (4/\lambda_j + \exp(\lambda_j))$ ).

For the analysis of the performance sensitivity of the distributed simulation protocols using the above model, the following parameters were considered:

- **Number of Available Processors.** With an increasing number of processors, more and more inherent model parallelism may be exploited, but interprocessor

communication costs also rise. At some point communication costs are expected to dominate the gain of parallel simulation work.

- **Initial Token Count.** The larger the number of tokens in one region (i.e., initial tokens in  $p_{in}$ ), the larger the model parallelism, i.e., the number of possibly concurrent executions of transition firings in different LPs. However, as the number of tokens in the region increases, so does the number of events which must be managed by SE, e.g., the average EVL size and access cost tends to grow.
- **Token Flow Rate among Regions.** By adjusting the ratio of the firing rates  $\lambda_{int} : \lambda_{ext}$ , it is possible to simulate the behavior of regions with varying degrees of “locality.” A large  $\lambda_{int} : \lambda_{ext}$  ratio simulates a model where the majority of the simulation steps affect only model parts within the LP’s own region, whereas a small ratio causes increased intra-LP token transfers, and thus communication overheads. We shall refer to the value of  $\lambda_{int} : \lambda_{ext}$  as the internal/external event ratio.
- **Forecast Method.** We use all the forecast methods to estimate the arrival time of the next message as described above.
- **Throttling Delay.** We consider three cases to delay the execution of a scheduled event. In the first case (*implicit throttling*), once the simulator decides not to execute the next event, control just loops back to reading the input buffers section, thus implicitly delaying its execution. In the two other cases (*explicit throttling*) the simulation is explicitly delayed for a certain amount of time: the delay is either set to the average amount of CPU time used to execute one event, or the maximum amount of CPU time used to execute one event.

#### 4.2 TW with Implicit Throttling

In the first set of experiments, the performance of TW with implicit throttling delays was investigated for  $N = 2^0, \dots, 2^5$  processors of a 132 node Meiko CS-2, each simulating 64/ $N$  regions of the kind in Fig. 9. As a performance measure the classical “event rate” (committed transition firings per unit CPU-time) was translated into a metric called the “acceleration factor,” which stands for the total number of committed transition firings per processor per unit time (transition firings that do not contribute useful simulation work, i.e., are subject to annihilation in a rollback situation, are not counted). Fig. 10a, 10b, and 10c compare the acceleration factor for an internal/external event ratio ranging from

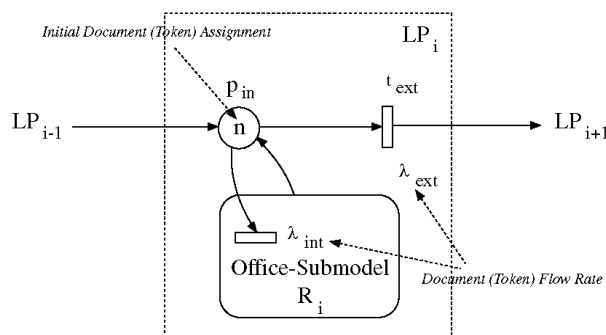


Fig. 9. Petri net region of a document flow system.

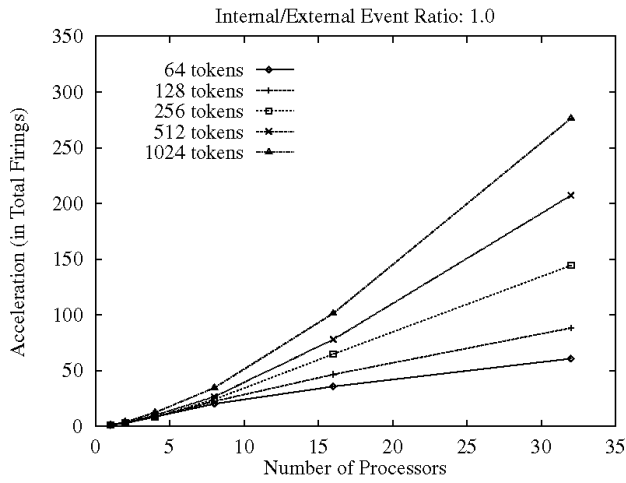
1 to 10,000 (Fig. 10d, 10e, and 10f). (All reported numbers were obtained by repeated execution runs guaranteeing statistical significance.) It is seen that for high model parallelism (a large number of initial tokens) the simulation engines using adaptive methods are able to execute 15 percent more transition firings (events) than a simulation engine using plain TW. In cases with a high degree of locality of event causalities ( $\lambda_{int}/\lambda_{ext} = 10K$ ), i.e., where causal effects of transition firings are restricted to the region of a single LP, the inter-LP communication overhead diminishes. Adaptive protocols in this case, also depending on the available model parallelism, are able to achieve better results for LPs with light (token) load, but perform worse in cases of heavy loaded LPs. This is because local event management overheads after a critical load becomes the prevalent CPU cost factor. Almost irrespective of the simulation protocol, if there are less initial tokens, simulation engines have less simulation work: the average length of EVL is shorter, and therefore state saving and possible rollbacks are performed faster. In our model with 32 processors the average event processing and state saving time is 3,981  $\mu\text{sec}$  in the 1,024 documents (tokens) case, and 2,872  $\mu\text{sec}$  in the 64 documents case. The average rollback cost even decreases from 105,153  $\mu\text{sec}$  to 1,024  $\mu\text{sec}$ , so that a more detailed investigation is demanded to explain when the adaptive approach is worthwhile.

For an analysis of the gain of the adaptive mechanisms we express the relation between the costs for rollbacks and the forecast method as:

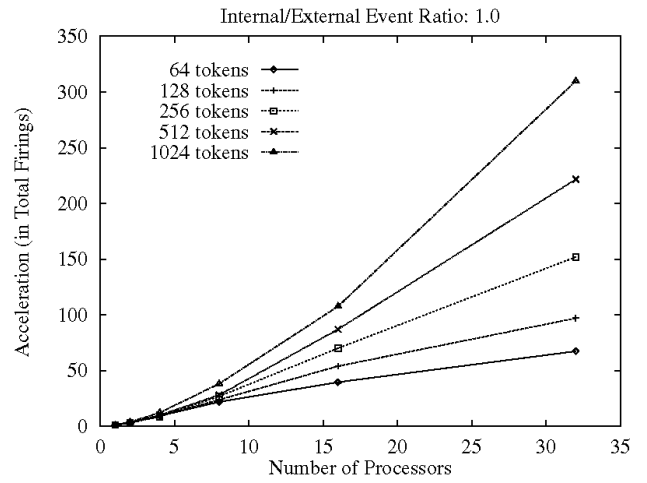
$$\sum_{n=0}^N t_{n, \text{Rollback}(TW)} \geq \sum_{n'=0}^{N'} t'_{n', \text{Rollback}(\text{Method})} + \sum_{m=0}^M t_{m, \text{Update}} + \sum_{i=0}^I t_{i, \text{EstimateMethods}}, \quad (6)$$

where  $N$  is the number of rollbacks in plain TW,  $t_{n, \text{Rollback}(TW)}$  is the time for performing the  $n$ th rollback,  $N'$  and  $t'_{n', \text{Rollback}(\text{Method})}$  are the respective parameters for adaptive TW.  $M$  is the number of external positive messages and  $I$  is the number of estimate function calls. Under normal conditions we can say, that  $M > N$  and  $I \gg M$ . The adaptive methods perform better (only) if the aggregated times on the right side of (6) are smaller than the sum of the rollback times in plain TW. In our examples this is the case at a “token load” of 512 and higher. Otherwise the computational forecast overhead dominates the gain.

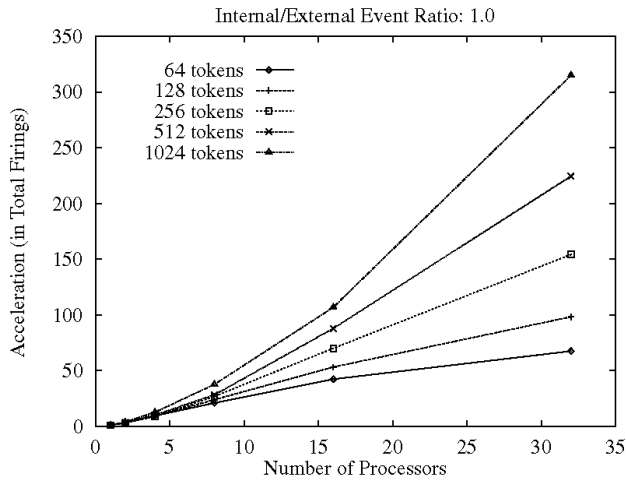
Fig. 11 shows a comparison of the aggregated CPU cost for rollback calls (Rollback) against the CPU time for forecast procedure calls (Forecast) and arrival history updates (Update) for subnets with an internal/external event ratio of 1 and 1,024 tokens (Fig. 11a) and 64 tokens (Fig. 11b)), mapped on 32 processors, using plain TW and TW with arithmetic mean forecast. Both cases reveal a reduction of the rollback cost, but in the loaded system (1,024 initial tokens) the rollback cost reduction can even overwhelm the forecast cost. In less loaded cases, (64 initial tokens) the rollback chains are shorter, so that the cumulated rollback overhead remains comparably small, and the overall performance gain from rollback cost reduction cannot hide the forecast overhead.



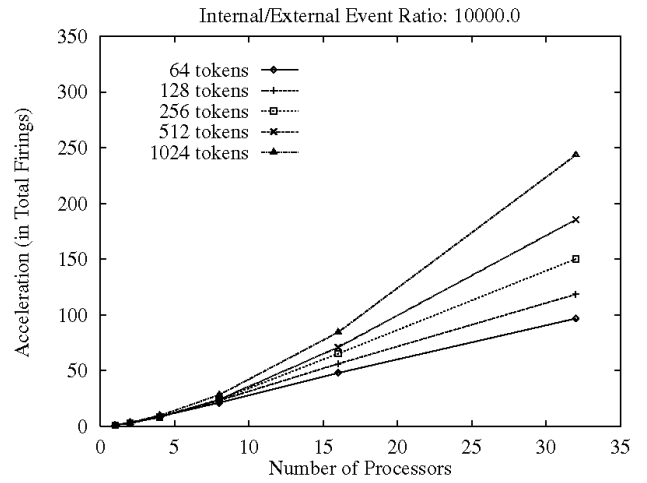
(a)



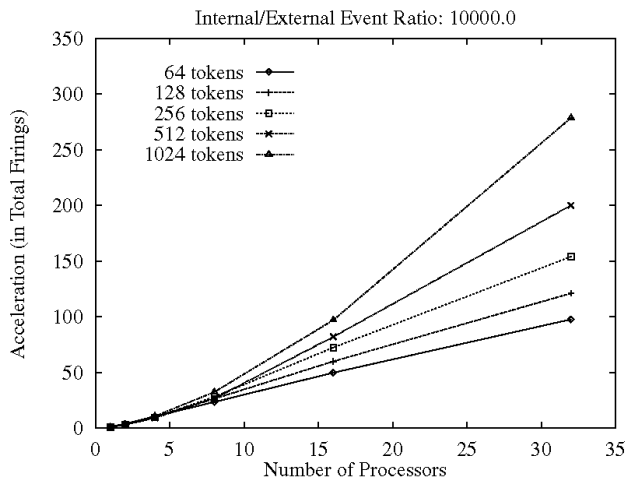
(b)



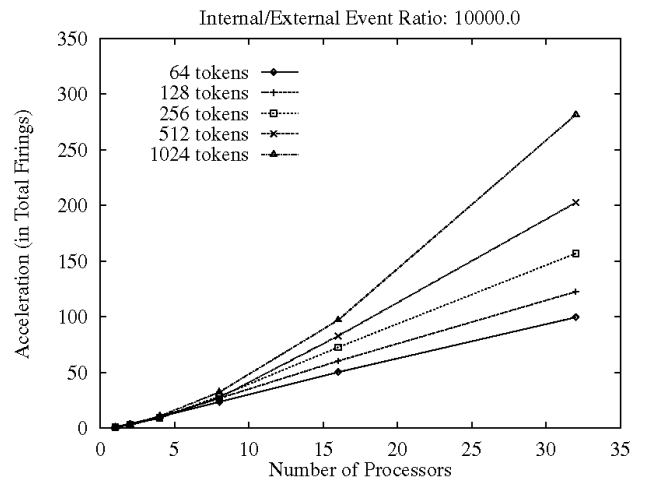
(c)



(d)



(e)



(f)

Fig. 10. Performance sensitivity with respect to internal/external event ratio (implicit throttling). (a) TW,  $\lambda_{int}/\lambda_{ext} = 1$ ; (b) TW + Exp. Smooth.  $\lambda_{int}/\lambda_{ext} = 1$ ; (c) TW + ARMA,  $\lambda_{int}/\lambda_{ext} = 1$ ; (d) TW,  $\lambda_{int}/\lambda_{ext} = 10K$ ; (e) TW + exp. smooth,  $\lambda_{int}/\lambda_{ext} = 10K$ ; (f) TW + ARMA,  $\lambda_{int}/\lambda_{ext} = 10K$ .

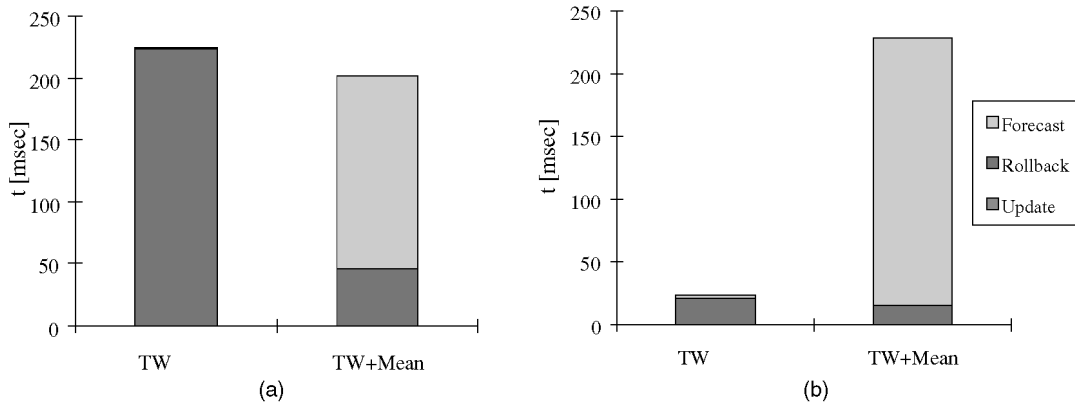


Fig. 11. Token load: (a) 1,024; (b) 64.

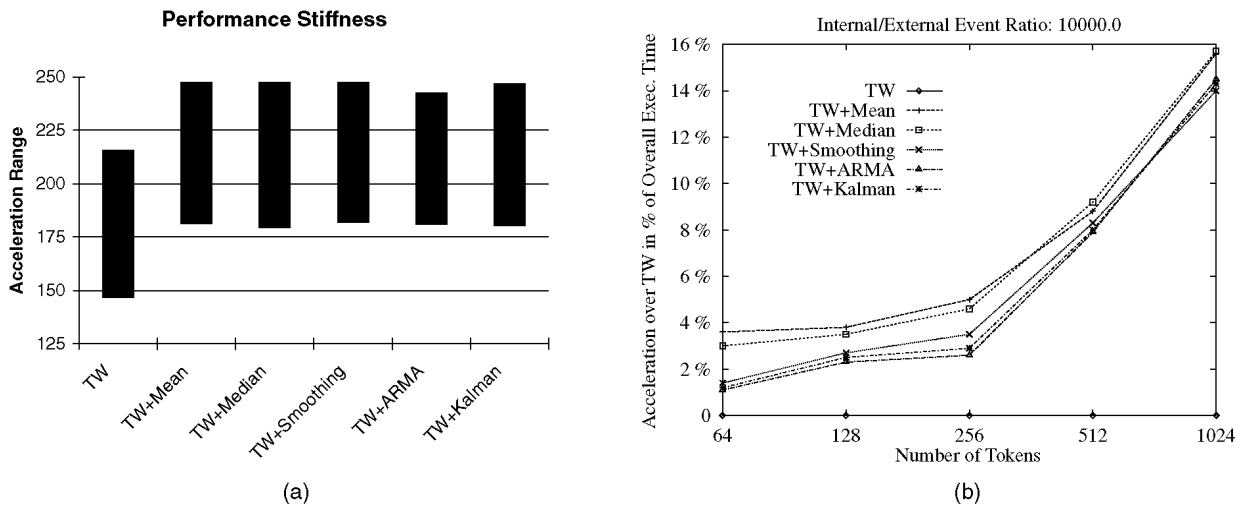


Fig. 12. Performance stiffness with respect to: (a) locality; (b) performance gain over plain TW.

Fig. 10, focusing on a comparison of plain TW against central tendency forecasts (represented by the exponential smoothing strategy) and the time series based forecast (represented by the ARMA strategy), also explains the sensitivity of the methods with respect to “locality” (internal/external event ratio). At a “level of locality” of  $\lambda_{int}/\lambda_{ext} = 1$ , depending on the initial load, all the strategies achieve a certain acceleration. Increasing the “level of locality” to  $\lambda_{int}/\lambda_{ext} = 10K$  all the strategies deliver worse best cases, but better worst cases at high processor numbers. This is a clear indication that models of high “level of locality” are stiffer with respect to performance, and holds also for the methods not presented in Fig. 10 (summarized in Fig. 12a) for the involvement of 32 processors (the plot for each forecast method explains the spread of acceleration attainable with  $\lambda_{int}/\lambda_{ext}$  ranging from 1 to 10K). A comparison of the relative accelerations as induced by the various forecast strategies (for the particular case of  $\lambda_{int}/\lambda_{ext} = 10K$  executing on 32 nodes of the CS-2) is summarized in Fig. 12b.

A second group of experiments considered deterministic transition timing. Because of the symmetric nature of the simulation model and its deterministic timing, the time gap between LVT and GVT within each LP appeared to be comparably small. In other words, the probability of rollback is small in this cases, and rollback costs are lower than with transitions with stochastic enablings. The average time

needed to perform a rollback in the 32 processor setting decreased from  $\approx 7$  msec to  $\approx 2$  msec (the simulation of one event still needs about 3 msec). The cost induced by the larger number of forecast calls (each about  $140 \mu\text{sec}$ ), exceeded the gain in rollback performance, and a 7 percent performance degrade over TW was observed. Finally, when transition timing was mixed, the performance gap between TW and the adaptive methods was not significant.

### 4.3 TW with Explicit Throttling

The previous section demonstrated that TW could be accelerated by just starting a new loop iteration (`continue;` in step (s9), Fig. 4), instead of simulating every event immediately. This loop back represents an implicit throttle since rather than executing the next scheduled event, the input buffers are inspected for new message arrivals instead. The implicit delay here corresponds to the CPU cost for handling the new arrivals, i.e., the number of received messages. (Note the possibility of this strategy to detect stragglers earlier, since the input buffers are polled more frequently.)

Another variation of throttling is to explicitly delay the execution of events in step (s9) of the adaptive TW engine (Fig. 4). Fig. 13 shows the acceleration gained when the CPU is delayed by  $3,000 \mu\text{sec}$  in step s9, before starting a new loop iteration (the average event processing time

varies between 2,000  $\mu\text{sec}$  and 4,000  $\mu\text{sec}$ , depending on the internal/external event ratio and the token load). Explicitly delaying yields an additional acceleration over TW (as in Fig. 13a, 13b, and 13c, but (slightly) degrades performance over the implicit throttling approach. An intuitive suggestion for the issue of finding an “appropriate” delay value is to adapt the explicit throttle to the load of an LP, since with a fixed explicit throttle, the blocking time is always either too short or too long. Experiments with a dynamic, periodic readjustments of the delay times (after  $n = 1, 10, 100$  transition firings) are also reported in Fig. 13d, 13e, and 13f. Again, the acceleration for a high token load is little better than for plain TW, but still worse than the acceleration in the case of implicit throttling.

In conclusion, for the explicit throttling approach we can thus say that for the particular type of simulation models in combination with our target execution platform, it is more important to poll the input buffers for stragglers more frequently, i.e., to detect potential rollbacks earlier, than to idle for possible future stragglers.

#### 4.4 A Stress Case for TW: Time Varying Arrivals

With the ARMA and Kalman filter approach the hope is to better cope with message arrival processes exhibiting time series characteristics than the central tendency based methods (arithmetic mean, median, exponential smoothing). The previous case studies did not reveal a significant difference among the two classes, since for the particular arrival processes the central tendency forecasts appeared “good enough” to balance the performance of the more sophisticated methods.

To demonstrate the particular abilities of the time series based methods, we study the performance of the simulation model in Fig. 14, comprising the two LPs  $LP_1$  and  $LP_2$ . Since TW performance is known to degrade in cases of imbalance in the LVT progression of different LPs, we have intentionally constructed a worst case scenario (Fig. 14): tokens in this model can circulate along the T-invariants (or cycles)  $(t_1, t_9, t_5, t_{10})$ ,  $(t_3, t_{11}, t_7, t_{12})$ ,  $(t_2, t_{11}, t_8, t_{10})$ ,  $(t_4, t_9, t_6, t_{12})$ ,  $(t_2, t_{11}, t_7, t_{12}, t_4, t_9, t_5, t_{10})$  and  $(t_1, t_9, t_6, t_{12}, t_3, t_{11}, t_8, t_{10})$ . For example, if the model executes along the cycle  $(t_1, t_9, t_5, t_{10})$ , then  $LP_2$  progresses LVT at a ten-fold higher rate than  $LP_1$  ( $\tau(t_9) \sim \exp(1.0)$ ,  $\tau(t_{10}) = 10.0$ ), and  $LP_1$  will permanently force  $LP_2$  to rollback. If the model executes along  $(t_3, t_{11}, t_7, t_{12})$ , the opposite is the case, with the only difference that the dominating timing component is not deterministic, but of stochastic nature ( $\tau(t_{11}) \sim \exp(0.1)$ ). A switching probability is introduced for the conflicting transitions  $(t_1, t_2)$ ,  $(t_3, t_4)$ , etc. to define  $(t_1, t_9, t_5, t_{10})$  and  $(t_3, t_{11}, t_7, t_{12})$  as *main cycles*, and to control the switching of tokens among main cycles.  $p = 0.95$  in Fig. 15 expresses a 5 percent “willingness” of tokens to switch main cycles at any of the decision points  $(t_1, t_2)$ ,  $(t_3, t_4)$ ,  $(t_5, t_6)$  and  $(t_7, t_8)$ . For the initial marking  $\mu^{(0)}(1) = 2$  the LVT progression in the LPs reveals three different *phases*:

- **Phase 1.** Two tokens on  $(t_1, t_5, t_9, t_{10})$ , ( $LP_1$  progresses LVT approx. 10 times faster, at stochastic increments)
- **Phase 2.** Two tokens on  $(t_3, t_7, t_{11}, t_{12})$ , ( $LP_2$  progresses LVT approximately 10 times faster, at deterministic increments) and

- **Phase 3.** One token on  $(t_1, t_5, t_9, t_{10})$  and 1 token on  $(t_3, t_7, t_{11}, t_{12})$ , (mixed stochastic/deterministic LVT progression).

It is seen that  $p = 0.75$  allows for more, and  $p = 0.95$  for less frequent cycle switching. Cycle switching of a token in the model causes the switching among the different execution phases, which exposes a “stress case” for TW in general, and the central tendency based adaptive TW protocols in particular. LVT progression traces as obtained on the CM-5 are depicted in Fig. 15a, which empirically verify our arguments. Fig. 15b shows that the ARMA approach is able to identify the different execution phases, i.e., an autoregressive phase (top) in a certain window at  $p = 0.95$ , as well as a moving average phase (bottom) in another window at  $p = 0.75$ .

The superiority of the ARMA based method in this stress case is summarized in Fig. 16, showing the amount of CPU-time consumed to progress virtual time to LVT = 1,000 in the model of Fig. 14: If the simulation model exhibits phases of different token throughput rates, plain TW can perform pathologically bad. This is even true for optimism throttling based on the arithmetic mean, specifically in phases where there is high variation in the timestamp increment carried by messages. For the particular example, TW with throttling based on ARMA models could outperform any other protocol, and accelerated TW by a factor of 2.

## 5 CONCLUSIONS

The quantitative analysis of time dynamic system represented in large Petri net performance models tends to become practically intractable with traditional evaluation techniques like analysis or sequential discrete event simulation. Parallel and/or distributed simulation mechanisms are demanded to cope with very complex and large models.

We have improved the standard Time Warp distributed simulation technique by introducing an adaptive optimism control mechanism. The adaptive protocol is dedicated to the distributed simulation of spatially decomposed timed Petri nets, where submodels (subnets) are simulated in logical processes which execute on individual processors in a multiprocessor environment. Our simulation engine, by temporarily blocking the processing of local transition firings, avoids the generation and sendout of token messages in states for which it is likely that they will have to be “rolled back,” thus potentially reducing rollback overhead costs. “On-the-fly” statistical analysis of the token message arrival history is used to make forecasts for the timestamps of future tokens, thus enabling every logical process to adapt its local synchronization behavior to the most efficient strategy with respect to the anticipated future.

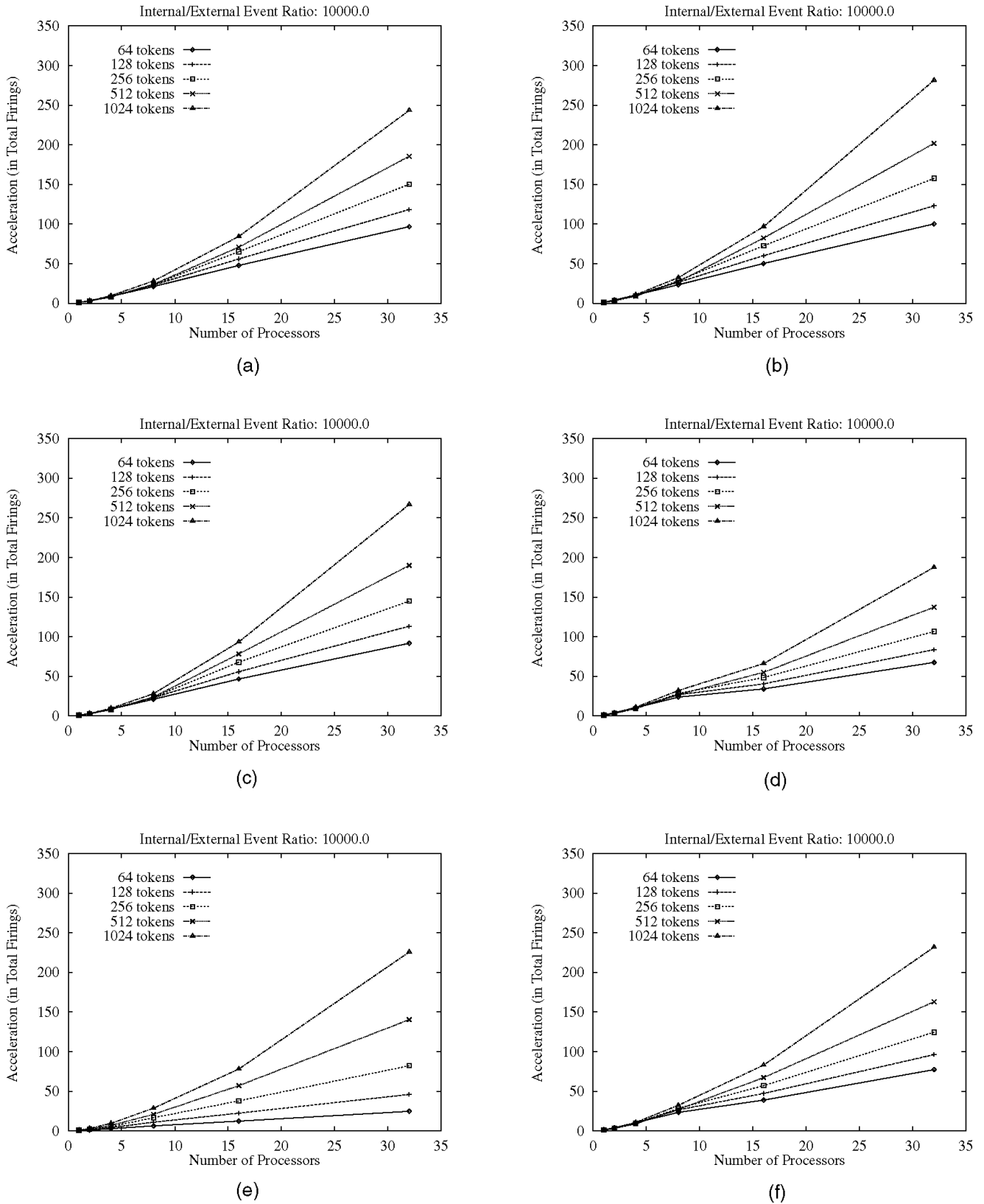


Fig. 13. Performance sensitivity with respect to throttling delay (explicit throttling). (a) TW; (b) implicit throttle; (c) explicit throttle 3,000  $\mu$ sec; (d) dynamic (readjust after 1); (e) dynamic (readjust after 10); (f) dynamic (readjust after 100).

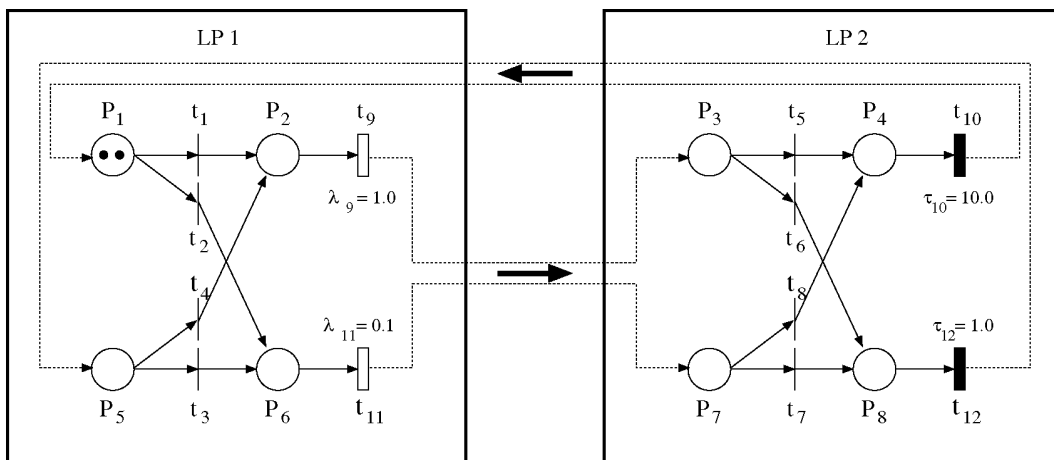


Fig. 14. Mixed timing model inducing LVT progression in phases.

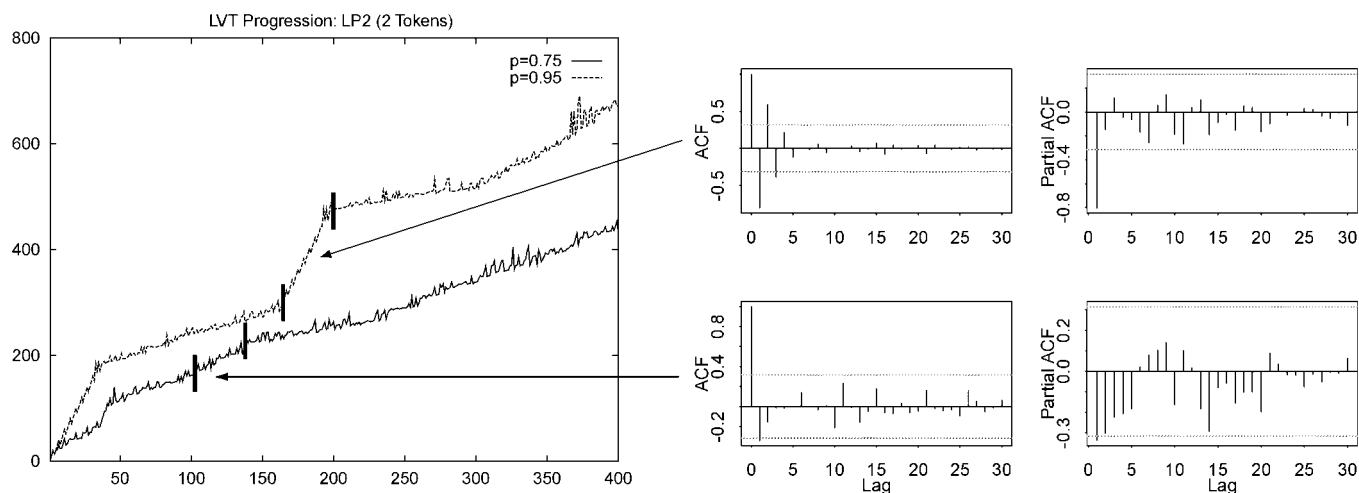


Fig. 15. Message No vs. message timestamp as observed on CM-5; ARMA characteristics.

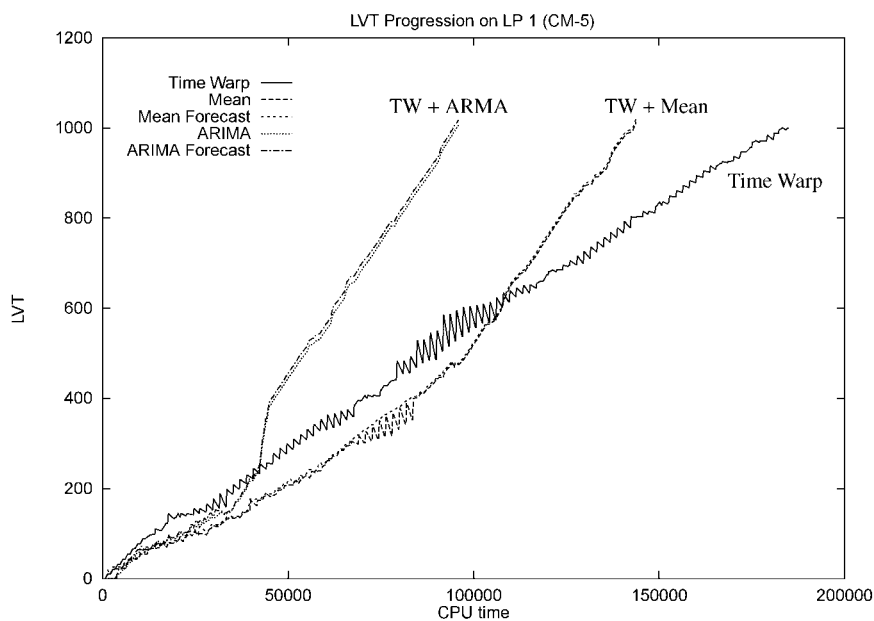


Fig. 16. LVT progression of plain TW, TW with mean forecast, and TW with ARMA forecast.

Two classes of forecast methods were studied:

- For estimates based on (weighted) means, efficient incremental procedures can be implemented for next message timestamp forecast, causing negligible or minor intrusion on the simulation engine. Those methods (arithmetic mean, median based and exponential smoothing), while improving Time Warp performance significantly in the absence of trends and seasons in the arrival process, cannot cope well with nonstationary arrivals.
- At the cost of higher computational complexity, time series based forecast methods with a more adequate predictions in the case of periodic and correlated channel time increments can be used. We have developed forecast methods based on autoregressive moving average (ARMA[ $p, q$ ]) models and on Kalman filters. These approaches can even further accelerate Time Warp for "loaded" systems, i.e., where the message load is comparably high. It can, however, also tend to slow down Time Warp in case there are not enough messages circulating among logical processes, such that either no statistical significance can be achieved for the established forecast model, or the computational complexity for forecasting overwhelms the overall execution time gain.

For the distributed Time Warp simulation of very large Petri net models we achieve dramatic performance improvements, e.g., a 250- to 300-fold acceleration of the overall execution speed using 32 processors on the Meiko CS-2 multiprocessor (as compared to having the whole simulation model executed by a single Time Warp logical process on a single processor). This effect mainly results from the decomposition of the Petri net model into smaller submodels which reduces memory access overhead at quadratic order. Empirical evidence has been derived that in cases where message arrival processes exhibit nonstationarity, i.e., logical processes progress local simulations in phases of varying performance behavior, the time series based methods significantly outperform central tendency based methods.

As an optimism control approach to Time Warp we find that probabilistic throttling gains adaptiveness in several respects:

- 1) Independent of the communication-computation speed ratio of the target execution platform, the synchronization policy is automatically adjusted to that point in the spectrum between the optimistic Time Warp and conservative Chandy/Misra/Bryant class of protocols, that is most appropriate for the inherent model parallelism. (This property makes the simulator portable to different execution platforms, avoiding costly performance tuning efforts.)
- 2) Logical processes are "self-adaptive," i.e., control decisions are conducted locally based on observations of the behavior of surrounding logical processes. (This property allows for an efficient exploitation of simulation model parallelism that is "local" to particular neighborhoods of regions of the (global) Petri net model.)
- 3) Even if LVT progresses in "phases," logical processes dynamically readjust their synchronization policy.

(This property, together with self-adaptiveness, in a natural way evades the partitioning problem under imbalanced loads.)

## APPENDIX A—TIME SERIES FORECASTING

Appendix A gives the statistical and algorithmic details for timestamp forecasting based on the consideration of  $(\delta_1, \delta_2, \dots, \delta_n)$  as realizations of random variables  $X_t, t \in T$ .

Generally when analyzing time series, a first insight into the dependencies among random variables is gained from covariance analysis. We briefly recall: If  $\{X_t, t \in T\}$  is a process such that  $\text{Var}(X_t) < \infty$  for each  $t \in T$ , then the **autocovariance function**  $\gamma_x(\cdot, \cdot)$  of  $\{X_t\}$  is defined by

$$\gamma_x(r, s) = \text{Cov}(X_r, X_s), \quad r, s \in T. \quad (7)$$

The process  $\{X_t, t \in \mathbb{Z}\}$  with index set  $\mathbb{Z} = \{0, \pm 1, \pm 2, \dots\}$  is said to be **stationary** if

- 1)  $E|X_t|^2 < \infty \quad \forall t \in \mathbb{Z}$ ,
- 2)  $EX_t = m \quad \forall t \in \mathbb{Z}$ ,

and

- 3)  $\gamma_x(r, s) = \gamma_x(r+t, s+t) \quad \forall r, s, t \in \mathbb{Z}$ .

If  $\{X_t, t \in \mathbb{Z}\}$  is stationary, then  $\gamma_x(r, s) = \gamma_x(r-s, 0)$  for all  $r, s, \in \mathbb{Z}$ . It is, therefore, convenient to redefine the autocovariance function of a stationary process as the function of just one variable:

$$\gamma_x(h) = \gamma_x(h, 0) = \text{Cov}(X_{t+h}, X_t) \quad t, h \in \mathbb{Z} \quad (8)$$

The function  $\gamma_x(\cdot)$  will be referred to as the *autocovariance function* of  $\{X_t\}$  and  $\gamma_x(h)$  as its value at "lag"  $h$ . The **autocorrelation function (acf)** of  $\{X_t\}$  is defined analogously as the function whose value at lag  $h$  is [6]

$$\rho(h) \equiv \gamma_x(h) / \gamma_x(0) = \text{Corr}(X_{t+h}, X_t) \quad t, h \in \mathbb{Z}. \quad (9)$$

If  $\gamma(\cdot)$  is the autocovariance function of a stationary process  $\{X_t, t \in \mathbb{Z}\}$ , then

$$\gamma(0) \geq 0, \quad (10)$$

$$|\gamma(h)| \leq \gamma(0) \quad \forall h \in \mathbb{Z} \quad (11)$$

and

$$\gamma(h) = \gamma(-h) \quad \forall h \in \mathbb{Z}. \quad (12)$$

From the observation  $\{x_1, x_2, \dots, x_n\}$  of a stationary time series  $\{X_t\}$  we have to estimate the autocovariance function  $\gamma(\cdot)$  of the underlying process  $\{X_t\}$  to be able to construct an appropriate model for the message arrival pattern. The estimate of  $\gamma(\cdot)$  which we shall use is the *sample autocovariance function*.

**DEFINITION A.1.** *The sample autocovariance function of  $\{x_1, x_2, \dots, x_n\}$  is defined [6]*

$$\hat{\gamma}(h) = \frac{1}{h} \sum_{j=1}^{n-h} (x_{j+h} - \bar{x})(x_j - \bar{x}), \quad 0 \leq h \leq n, \quad (13)$$

and  $\hat{\gamma}(h) = \hat{\gamma}(-h)$ ,  $-n \leq h \leq 0$ , where  $\bar{x}$  is the sample mean  $\bar{x} = n^{-1} \sum_{j=1}^n x_j$ .

The divisor  $n$  is used rather than  $(n - h)$  since this ensures that the matrix  $\hat{\Gamma}_n = [\hat{\gamma}(i - j)]_{i,j=1}^n$  is nonnegative definite. In the future we will only consider centered time series  $\{X_t\}$  ( $\bar{x} = 0$ ) (i.e., transformed with respect to the series mean  $x_i = \delta_i - \mu$ ). The **partial autocorrelation** (pacf), like the autocorrelation function, expresses essential information about the dependence structure of the stationary process. The partial autocorrelation  $\alpha(k)$  at lag  $k$  may be regarded as the correlation between  $X_1$  and  $X_{k+1}$ , after removing the effect of the intervening observation  $X_2, \dots, X_k$ . The pacf is obtained from [6]

$$\begin{bmatrix} \rho(0) & \rho(1) & \dots & \rho(k-1) \\ \rho(1) & \rho(2) & \dots & \rho(k-2) \\ \vdots & \vdots & \ddots & \vdots \\ \rho(k-1) & \rho(k-2) & \dots & \rho(0) \end{bmatrix} \begin{bmatrix} \phi_{k1} \\ \phi_{k2} \\ \vdots \\ \phi_{kk} \end{bmatrix} = \begin{bmatrix} \rho(1) \\ \rho(2) \\ \vdots \\ \rho(k) \end{bmatrix}, \quad k \geq 1 \quad (14)$$

The partial autocorrelation  $\alpha(k)$  of  $\{X_t\}$  at lag  $k$  is  $\alpha(k) = \phi_{kk}$  for  $k > 1$ , where  $\phi_{kk}$  is uniquely determined by (14).

### A.1 Stationary ARMA Processes

Since the 1970 work by Box and Jenkins [5] autoregressive moving average (ARMA) models have become popular and important tool in the modeling and forecasting of time series data.

**DEFINITION A.2.** (The ARMA( $p, q$ ) Process). *The process  $\{X_t, t = 0, \pm 1, \pm 2, \dots\}$  is said to be an ARMA( $p, q$ ) process if  $\{X_t\}$  is stationary and for every  $t$ ,*

$$X_t - \phi_1 X_{t-1} - \dots - \phi_p X_{t-p} = Z_t = \theta_1 Z_{t-1} - \dots - \theta_q Z_{t-q} \quad (15)$$

where  $\{Z_t\} \sim WN(0, \sigma^2)$ .

Using the backward shift operator, (15) can be written in the more compact form

$$\phi(B)X_t = \theta(B)Z_t, \quad t = 0, \pm 1, \pm 2, \dots, \quad (16)$$

where  $\phi$  and  $\theta$  are polynomials at degree  $p$  and  $q$ , respectively:

$$\phi(z) = 1 - \phi_1 z - \dots - \phi_p z^p \quad (17)$$

$$\theta(z) = 1 + \theta_1 z + \dots + \theta_q z^q, \quad (18)$$

and the backward shift operator  $B$  is defined by

$$B^j X_t = X_{t-j}, \quad j = 0, \pm 1, \pm 2, \dots \quad (19)$$

A process is called a *moving average process* of order  $q$  (denoted as MA( $q$ )), if  $\phi(z) \equiv 1$ :

$$X_t = \theta(B)Z_t \quad (20)$$

A realization of  $\{X_1, \dots, X_{90}\}$  of an MA(2)  $X_t = 0.7Z_{t-1} - 0.2Z_{t-2} + WN(0, 1)$  is shown in Fig. 17a.

A process is called an *autoregressive process* of order  $p$  (denoted as AR( $p$ )), if  $\theta(z) \equiv 1$ :

$$\phi(B)X_t = Z_t \quad (21)$$

As an example, a realization  $\{X_1, \dots, X_{90}\}$  of an AR(2)  $X_t = 0.9X_{t-1} - 0.2X_{t-2} + WN(0, 1)$  is shown in Fig. 17b.

It can be shown that for every model (15) a representation exists as either an **infinite AR** (AR( $\infty$ )), or an **infinite MA** (MA( $\infty$ )) process. The model can be written either as

$$\sum_{j=0}^{\infty} \pi_j X_{t-j} = \pi(B)X_t = Z_t \quad (22)$$

or as

$$X_t = \sum_{j=0}^{\infty} \psi_j Z_{t-j} = \psi(B)Z_t \quad (23)$$

where

$$\pi(z) = \sum_{j=0}^{\infty} \pi_j z^j = \phi(z) / \theta(z) \quad |z| \leq 1 \quad (24)$$

$$\psi(z) = \sum_{j=0}^{\infty} \psi_j z^j = \phi(z) / \theta(z) \quad |z| \leq 1 \quad (25)$$

are an infinite series of constants  $\{\pi_j\}$  ( $\{\psi_j\}$ ) such that  $\sum_{j=0}^{\infty} |(\sum_{j=0}^{\infty} |\psi_j|) < \infty$ .

### A.2 Prediction of Stationary Processes

The idea to predict the values  $\{X_t, t \geq n+1\}$  of a stationary timestamp increment process based on  $\{X_1, \dots, X_n\}$  is to use the observations of this process made so far. Besides the direct computation of the predictors, where a large system of linear equations has to be solved [6], a recursive method exists for the one-step predictors  $X_{n+1}$ ,  $n \geq 1$ . Given the observations from a zero-mean stationary time series, with  $\hat{\gamma}(0) > 0$ , we can fit an autoregressive process of order  $m < n$

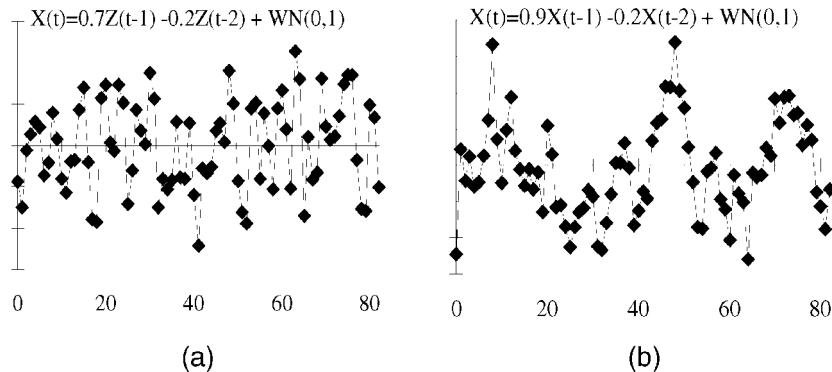


Fig. 17. A moving average: (a) (MA(2)) process; (b) an autoregressive (AR(2)) process.

to the data using the *Durbin-Levinson* algorithm. The fitted AR( $m$ ) process in this case is

$$X_t - \hat{\phi}_{m1}X_{t-1} - \dots - \hat{\phi}_{mm}X_{t-m} = Z_t \{Z_t\} \sim WN(0, \hat{v}_m) \quad (26)$$

which can be determined for  $m = 1, 2, \dots, n-1$  recursively from the relations,  $\hat{\phi}_{11} = \hat{\rho}(1)$ ,  $\hat{v}_0 = \hat{\gamma}(0)$ ,

$$\hat{\phi}_{mm} = \left[ \hat{\lambda}(m) - \sum_{j=1}^{m-1} \hat{\phi}_{m-1,j} \hat{\lambda}(m-j) \right] / \hat{v}_{m-1}, \quad (27)$$

$$\begin{bmatrix} \hat{\phi}_{m1} \\ \hat{\phi}_{m2} \\ \vdots \\ \hat{\phi}_{m,m-1} \end{bmatrix} = \begin{bmatrix} \hat{\phi}_{m-1,1} \\ \hat{\phi}_{m-1,2} \\ \vdots \\ \hat{\phi}_{m-1,m-1} \end{bmatrix} - \hat{\phi}_{m-1,m} = \begin{bmatrix} \hat{\phi}_{m-1,m-1} \\ \hat{\phi}_{m-1,m-2} \\ \vdots \\ \hat{\phi}_{m-1,1} \end{bmatrix} \quad (28)$$

$$\hat{v}_m = \hat{v}_{m-1} (1 - \hat{\phi}_{mm}^2), \quad (29)$$

where the  $\hat{\phi}_{11}, \hat{\phi}_{22}, \dots$  are estimates for the partial autocorrelation  $\hat{\alpha}$  at lags 1, 2, ... .

Unlike the Durbin-Levinson Algorithm for fitting autoregressive models, there is a recursive way of fitting moving average models

$$X_t = Z_t - \hat{\theta}_{m1}Z_{t-1} - \dots - \hat{\theta}_{mm}Z_{t-m} \{Z_t\} \sim WN(0, \hat{v}_m) \quad (30)$$

of orders  $m = 1, 2, \dots$  by means of the *innovations algorithm*. The estimates of  $\hat{\theta}_m$  and white noise variance  $\hat{v}_m$ ,  $m = 1, 2, \dots$  are obtained as follows: If  $\hat{\gamma}(0) > 0$ , we define the estimates  $\hat{\theta}$  and  $\hat{v}_m$  in (30) for  $m = 1, 2, \dots$ , by the recursion,  $\hat{v}_0 = \hat{\gamma}(0)$ ,

$$\hat{\theta}_{m,m-k} = \frac{1}{\hat{v}_k} \left[ \hat{\gamma}(m-k) - \sum_{j=0}^{k-1} \hat{\theta}_{m,m-j} \hat{\theta}_{k,k-j} \hat{v}_j \right], \quad (31)$$

$$k = 0, \dots, m-1$$

and

$$\hat{v}_m = \hat{\gamma}(0) - \sum_{j=0}^{m-1} \hat{\theta}_{m,m-j}^2 \hat{v}_j. \quad (32)$$

**Preliminary Estimation for ARMA( $p, q$ ) Processes.** If  $\{X_t\}$  is a zero-mean ARMA( $p, q$ ) process,

$$X_t - \phi_1 X_{t-1} - \dots - \phi_p X_{t-p} = Z_t - \theta_1 Z_{t-1} - \dots - \theta_q Z_{t-q} \quad (33)$$

$$\{Z_t\} \sim WN(0, \sigma^2)$$

then (5) can be written as an infinite MA model

$$X_t = \sum_{j=0}^{\infty} \psi_j Z_{t-j}$$

We can also rewrite (25), such that the coefficients  $\psi_j$  satisfy

$$\begin{cases} \psi_0 = 1, \\ \psi_j = \theta_j + \sum_{i=1}^{\min(j,p)} \phi_i \psi_{j-i}, \quad j = 1, 2, \dots \end{cases} \quad (34)$$

and by convention  $\theta_j = 0$  for  $j > q$  and  $\phi_j = 0$  for  $j > p$ . To estimate  $\psi_1, \dots, \psi_{p+q}$  we can use the innovation algorithm estimates  $\hat{\theta}_{m1}, \dots, \hat{\theta}_{m,p+q}$  of an MA( $m$ ) ( $p+q \ll m \approx n^{1/3}$ ). Re-

placing  $\psi_j$  by  $\hat{\theta}_{mj}$  in (34) and solving the resulting equations,

$$\hat{\theta}_{mj} = \theta_j + \sum_{i=1}^{\min(j,p)} \phi_i \hat{\theta}_{m,j-i}, \quad j = 1, 2, \dots, p+q, \quad (35)$$

we obtain initial parameter estimates for  $\phi_i$  ( $1 \leq i \leq p$ ) and  $\theta_j$  ( $1 \leq j \leq q$ ). From (35) with  $j = q+1, \dots, q+p$ , we see that  $\hat{\phi}_i$  should satisfy the equation:

$$\begin{bmatrix} \hat{\phi}_{m,q+1} \\ \hat{\phi}_{m,q+2} \\ \vdots \\ \hat{\phi}_{m,q+p} \end{bmatrix} = \begin{bmatrix} \hat{\phi}_{m,q} & \hat{\phi}_{m,q-1} & \dots & \hat{\phi}_{m,q+1-p} \\ \hat{\phi}_{m,q+1} & \hat{\phi}_{m,q} & \dots & \hat{\phi}_{m,q+2-p} \\ \vdots & \vdots & \dots & \vdots \\ \hat{\phi}_{m,q+p-1} & \hat{\phi}_{m,q+p-2} & \dots & \hat{\phi}_{m,q} \end{bmatrix} \begin{bmatrix} \hat{\phi}_1 \\ \hat{\phi}_2 \\ \vdots \\ \hat{\phi}_p \end{bmatrix} \quad (36)$$

After solving (36) for  $\phi_i$  ( $1 \leq i \leq p$ ), the estimates of  $\theta_j$  ( $1 \leq j \leq q$ ) are obtained from

$$\hat{\theta}_j = \theta_{mj} - \sum_{i=1}^{\min(j,p)} \hat{\phi}_i \hat{\theta}_{m,j-i}, \quad j = 1, 2, \dots, q \quad (37)$$

The white noise variance  $\sigma^2$  is estimate by  $\sigma^{\hat{v}} = \hat{v}_m$ .

### Maximum Likelihood Estimation for ARMA Processes.

The preliminary estimates of  $\theta_j$  and  $\phi_j$  are now used as an initial start value for the nonlinear optimization carried out in the course of maximizing the maximum likelihood. The one-step predictor  $\hat{X}_{n+1}$  of an ARMA( $p, q$ ) is given by

$$\hat{X}_{n+1} = \begin{cases} \sum_{j=1}^n \theta_{nj} (X_{n+1-j} - \hat{X}_{n+1-j}), & 1 \leq i < m = \max(p, q), \\ \sum_{j=1}^p \phi_j X_{n+1-j} + \sum_{j=1}^q \theta_{nj} (X_{n+1-j} - \hat{X}_{n+1-j}), & i \geq m. \end{cases} \quad (38)$$

and the likelihood of the observation  $(X_1, X_2, \dots, X_n)$  is

$$L(\bar{\theta}, \bar{\phi}, \sigma^2) = (2\pi)^{-n/2} (v_0 v_1 \dots v_{n-1})^{-1/2} * \exp \left[ -\frac{1}{2} \sum_{j=1}^n (X_j - \hat{X}_j)^2 / v_{j-1} \right]. \quad (39)$$

Criteria have been developed (e.g., Akaike's BIC and AIC criterion) attempting to prevent from overfitting by effectively assigning a cost to the introduction of each additional parameter. We use the AICC criterion defined by:

$$AICC(\bar{\theta}, \bar{\phi}) = -2 \ln L(\bar{\theta}, \bar{\phi}, \sigma^2) + \frac{2n(p+q+1)}{(n-p-q-2)} \quad (40)$$

### A.3 Identification Techniques

The problem of identifying the appropriate ARMA( $p, q$ ) model order to represent the time series  $\{X_t\}$  is to determine  $p$  and  $q$ . For an pure moving average process order identification is simple: the autocovariance function of the MA( $q$ ) process

$$X_t = \sum_{j=0}^q \theta_j Z_{t-j},$$

has the form (as depicted in Fig. 18a):

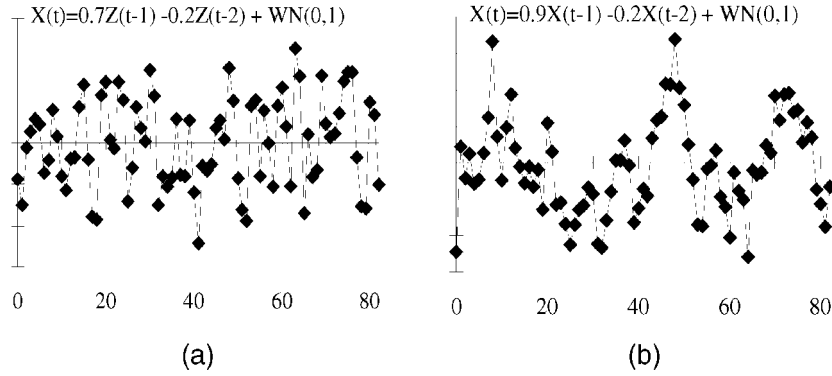


Fig. 18. Autocorrelation of: (a) an MA(2)-process; (b) an AR(2) process.

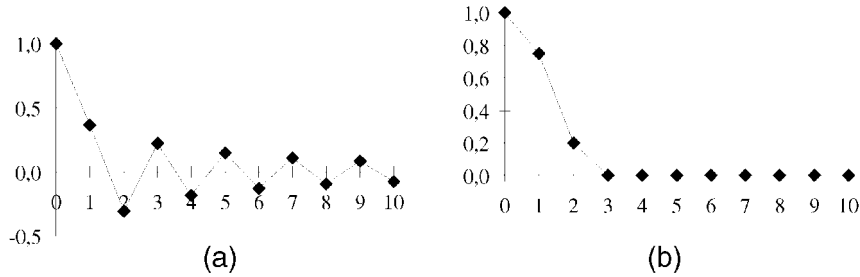


Fig. 19. Partial autocorrelation of: (a) an MA(2)-process; (b) an AR(2) process.

$$\gamma(k) = \begin{cases} \sigma^2 \sum_{j=0}^q \theta_j \theta_{j+|k|}, & |k| \leq q, \\ 0, & |k| > q, \end{cases} \quad (41)$$

where  $\theta_0$  is defined to be 1 and  $\theta_j, j > q$ , is defined to be zero.

For an pure autoregressive process AR( $p$ ) the order  $p$  can be obtained from the cut off in the partial autocorrelation function  $\alpha(\cdot)$  (see Fig. 19b):  $p$  could be easily determined as we know that

$$\alpha(k) = \begin{cases} \phi_{kk} & |k| \leq p, \\ 0, & |k| > p. \end{cases} \quad (42)$$

In contrast to the partial autocorrelation function of an AR( $p$ ) process that of an MA( $q$ ) process does not vanish for large lags. It is however bounded in absolute value by a geometrically decreasing function [6] (see Fig. 19a).

For fitting an AR( $p$ ) or MA( $q$ ) to an observed sample, the sample (partial-)autocorrelation  $\alpha(m)$  or  $\gamma(m)$  is not zero for  $m > p$  or  $q$ . If the order of the process is  $p$  or  $q$ , then for  $m > p$ ,  $\alpha(m)$  or  $m > q$ ,  $\gamma(m)$  will fall between the bounds  $\pm 1.96n^{-1/2}$ . If the underlying process is ARMA( $p, q$ ), then the model order identification requires the investigation of all meaningful pairs ( $p, q$ ) and choose that pair which minimizes one of the criteria given above. Pragmatically we first compute the AICC criterion with the preliminary estimates obtained by (36) and (37) for ARMA( $p_0, p_0$ ) models with  $p_0 = 1, 2, \dots$ . Then we try to eliminate one or more coefficients of the ARMA( $p_0 = p, p_0 = q$ )-model in order to minimize AICC.

## APPENDIX B—KALMAN FILTER FORECASTING

This method assumes that the series of interest  $\{X_t\}$  is not observed directly, but only as component in the random regression model

$$y_t = M\bar{x}_t + v \quad t = 1, 2, \dots, n \quad (43)$$

where  $M$  is a known  $1 \times p$  design vector which expresses the pattern that converts the unobserved stochastic vector  $\bar{x}_t$  into the observed series  $y_t$ .  $v$  is a zero-mean normally distributed noise vector. The random series  $\bar{x}_t$  is modeled as a multivariate process of the form

$$\bar{x}_t = \Phi\bar{x}_{t-1} + \bar{w}_t \quad t = 1, 2, \dots, n \quad (44)$$

where  $\Phi$  is a  $p \times p$  transition matrix describing the way the underlying series moves through successive time periods.  $\{X_t\}$  does not need to be stationary. The initial value  $\bar{x}_0$  is assumed to be a normal random vector with the  $p \times p$  covariance matrix  $\Sigma$ .  $\bar{w}_t$  is a  $p \times 1$  noise vector with zero-mean uncorrelated normal distributed terms and with common covariance matrix  $Q$ .

The model defined by (43) and (44) is a generalization of the ordinary AR( $p$ ) model in (21). A recursive method using the EM (*Expectation Maximization*) algorithm described in (e.g., [12]) has been proposed by Shumway and Stoffer [30] to compute the Kalman filter estimators, which is explained in the sequel.

The log likelihood of the complete data  $\bar{x}_0, \bar{x}_1, \dots, \bar{x}_n, \bar{y}_1, \dots, \bar{y}_n$  can be written in the form

$$\begin{aligned} \log L = & -\frac{1}{2} \log |\Sigma| - \frac{1}{2} (\bar{x}_0)' \Sigma^{-1} (\bar{x}_0) \\ & - \frac{n}{2} \log |Q| - \frac{1}{2} \sum_{t=1}^n (\bar{x}_t - \Phi\bar{x}_{t-1})' Q^{-1} (\bar{x}_t - \Phi\bar{x}_{t-1}) \\ & - \frac{n}{2} \log |V| - \frac{1}{2} \sum_{t=1}^n (\bar{y}_t - M\bar{x}_t)' V^{-1} (\bar{y}_t - M\bar{x}_t) \end{aligned} \quad (45)$$

The log likelihood given in (46) depends on the unobserved data series  $\bar{x}_0, \bar{x}_1, \dots, \bar{x}_n$ . We apply the EM algorithm on the observed series  $\bar{y}_0, \bar{y}_1, \dots, \bar{y}_n$  to maximize the log likelihood with respect to the parameters  $\Sigma, \Phi, Q, v$ . That is, determine the estimated parameters in iteration  $(r+1)$  as the values  $\Sigma, \Phi, Q$ , and  $v$  which maximize

$$G(\Sigma, \Phi, Q, v) = E_r(\log L|\bar{y}_0, \bar{y}_1, \dots, \bar{y}_n) \quad (46)$$

where  $E_r$  denotes the conditional expectation relative to a density containing the  $r$ th iteration values  $\Sigma_{(r)}, \Phi_{(r)}, Q_{(r)}$ , and  $v_{(r)}$  [30]. Rewriting (46) using

$$\bar{x}_t^n = E(\bar{x}_t|\bar{y}_1, \dots, \bar{y}_n), \quad (47)$$

and the covariance functions

$$P_t^n = \text{cov}(\bar{x}_t|\bar{y}_1, \dots, \bar{y}_n) \quad (48)$$

and

$$P_{t,t-1}^n = \text{cov}(\bar{x}_t, \bar{x}_{t-1}|\bar{y}_1, \dots, \bar{y}_n) \quad (49)$$

yields

$$\begin{aligned} G(\Sigma, \Phi, Q, v) = & -\frac{1}{2} \log |\Sigma| - \frac{1}{2} \text{trace}\{\Sigma^{-1}(P_0^n + \bar{x}_0^n)(\bar{x}_0^n)'\} \\ & -\frac{n}{2} \log |Q| - \frac{1}{2} \text{trace}\{Q^{-1}(C - B\Phi' - \Phi B' + \Phi A\Phi')\} \quad (50) \\ & -\frac{n}{2} \log |v| \\ & -\frac{1}{2} \text{trace}\left\{\frac{1}{v} \sum_{t=1}^n [(\bar{y}_t - M\bar{x}_t^n)(\bar{y}_t - M\bar{x}_t^n)' + MP_t^n M']\right\} \end{aligned}$$

where

$$A = \sum_{t=1}^n (P_{t-1}^n + \bar{x}_{t-1}^n \bar{x}_{t-1}^n)' \quad (51)$$

$$B = \sum_{t=1}^n (P_{t,t-1}^n + \bar{x}_{t-1}^n \bar{x}_{t-1}^n)' \quad (52)$$

and

$$C = \sum_{t=1}^n (P_t^n + \bar{x}_t^n \bar{x}_t^n)' \quad (53)$$

The Kalman filter terms  $\bar{x}_t^n, P_t^n$ , and  $P_{t,t-1}^n$  are computed under the parameter values of  $\Phi_{(r)}, Q_{(r)}$ , and  $v_{(r)}$  ( $\Sigma$  is fixed at a reasonable baseline level) using the recursions given by [23, pp. 201, 217] and [30, pp. 263].

For  $t = 1, \dots, n$

$$P_t^{t-1} = \Phi_{(r)} P_{t-1}^{t-1} \Phi_{(r)}' + Q_{(r)} \quad (54)$$

$$K_t = P_t^{t-1} M'(MP_t^{t-1} M' + v_{(r)})^{-1} \quad (55)$$

$$P_t^t = P_t^{t-1} - K_t MP_t^{t-1} \quad (56)$$

$$\bar{x}_t^{t-1} = \Phi_{(r)} \bar{x}_{t-1}^{t-1} \quad (57)$$

$$\bar{x}_t^t = \bar{x}_t^{t-1} + K_t(\bar{y}_t - M\bar{x}_t^{t-1}) \quad (58)$$

where we take  $\bar{x}_0^0 = \bar{0}$  and  $P_0^0 = \Sigma$ . In order to calculate  $\bar{x}_t^n$  and  $P_t^n$  one performs the set of backward recursions (for  $t = n, n-1, \dots, 1$ ) on the equations

$$J_{t-1} = P_{t-1}^{t-1} \Phi_{(r)}' (P_t^{t-1})^{-1} \quad (59)$$

$$\bar{x}_{t-1}^n = \bar{x}_{t-1}^{t-1} + J_{t-1}(\bar{x}_t^n) - \Phi_{(r)} \bar{x}_{t-1}^{t-1} \quad (60)$$

$$P_{t-1}^n = P_{t-1}^{t-1} + J_{t-1}(P_t^n - P_t^{t-1})J_{t-1}' \quad (61)$$

and  $P_{t,t-1}^n$  can be calculated using the backward recursion (for  $t = n, n-1, \dots, 2$ ) where

$$P_{n,n-1}^n = (I - K_n M) \Phi_{(r)} P_{n-1}^{n-1} \quad (62)$$

$$P_{t-1,t-2}^n = P_{t-1}^{t-1} J_{t-2}' + J_{t-1}(P_{t,t-1}^n - \Phi_{(r)} P_{t-1}^{t-1})J_{t-2}' \quad (63)$$

We get the next iteration of  $\Phi_{(r+1)}, Q_{(r+1)}$ , and  $v_{(r+1)}$  if we set

$$\Phi_{(r+1)} = BA^{-1}, \quad (64)$$

$$Q_{(r+1)} = n^{-1}(C - BA^{-1}B') \quad (65)$$

and

$$v_{(r+1)} = \frac{1}{n} \sum_{t=1}^n [(\bar{y}_t - M\bar{x}_t^n)(\bar{y}_t - M\bar{x}_t^n)' + MP_t^n M'] \quad (66)$$

which maximize the last two lines in the likelihood function (51).

Now the vector  $\bar{x}_t^n$  for  $t > n$  is the forecast value for the underlying series.

## ACKNOWLEDGMENTS

An earlier version of this paper appeared in the *Proceedings of the Seventh International Workshop on Petri Nets and Performance Models (PNPM'97)*, June 1997, Saint-Malo, France, pp. 205-216, IEEE CS Press, 1997.

## REFERENCES

- [1] H.H. Ammar and S. Deng, "Time Warp Simulation of Stochastic Petri Nets," *Proc. Fourth Int'l Workshop Petri Nets and Performance Models*, pp. 186-195, Los Alamitos, Calif.: IEEE CS Press, 1991.
- [2] F. Baccelli and M. Canales, "Parallel Simulation of Stochastic Petri Nets Using Recurrence Equations," *ACM Trans. Modeling and Computer Simulation*, vol. 3, no. 1, pp. 20-41, Jan. 1993.
- [3] F. Baccelli, N. Furmento, and B. Gaujal, "Parallel and Distributed Simulation of Free Choice Petri Nets," *Proc. Ninth Workshop Parallel and Distributed Simulation (PADS'95)*, pp. 3-10, 1995.
- [4] D. Ball and S. Hoyt, "The Adaptive Time-Warp Concurrency Control Algorithm," D. Nicol, ed., *Proc. SCS Multiconf. Distributed Simulation*, pp. 174-177, San Diego, Calif., Soc. for Computer Simulation. Simulation Series, vol. 22, no. 1, 1990.
- [5] G.E.P. Box and G.M. Jenkins, *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day, 1976.
- [6] P.J. Brockwell and R.A. Davis, *Time Series: Theory and Methods*. New York: Springer-Verlag, 1991.
- [7] G. Chiola and A. Ferscha, "Distributed Simulation of Petri Nets," *IEEE Parallel and Distributed Technology*, vol. 1, no. 3, pp. 33-50, Aug. 1993.
- [8] G. Chiola and A. Ferscha, "Distributed Simulation of Timed Petri Nets: Exploiting the Net Structure to Obtain Efficiency," M.A. Marsan, ed., *Proc. 14th Int'l Conf. Application and Theory of Petri Nets*, pp. 146-165, Chicago, June 1993, Lecture Notes in Computer Science 691, Berlin: Springer-Verlag, 1993.
- [9] M.Q. Cui and St. J. Turner, "A New Approach to the Distributed Simulation of Timed Petri Nets," A. Javor, A. Lehmann, and I.

- Molnar, eds., *Proc. 10th European Simulation Multiconf.*, pp. 90–94. SCS, 1996.
- [10] S.R. Das, "Adaptive Protocols for Parallel Discrete Event Simulation," J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, eds., *Proc. Winter Simulation Conf.*, pp. 186–193, 1996.
- [11] S.R. Das and R.M. Fujimoto, "An Adaptive Memory Management Protocol for Time Warp Parallel Simulation," *Proc. ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, pp. 201–210, Nashville, ACM, 1994.
- [12] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J.R. Statistical Soc.*, no. 39, pp. 1–38, 1977.
- [13] A. Ferscha, "Concurrent Execution of Timed Petri Nets," J.D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila, eds., *Proc. Winter Simulation Conf.*, pp. 229–236, 1994.
- [14] A. Ferscha, "Probabilistic Adaptive Direct Optimism Control in Time Warp," *Proc. Ninth Workshop Parallel and Distributed Simulation (PAD '95)*, pp. 120–129, 1995.
- [15] A. Ferscha, "Parallel and Distributed Simulation of Discrete Event Systems," A.Y. Zomaya, ed., *Parallel and Distributed Computing Handbook*, pp. 1,003–1,041. McGraw-Hill, 1996.
- [16] A. Ferscha, "Optimistic Distributed Execution of Business Process Models," *Proc. HICSS-31*, pp. 723–732, IEEE CS Press, 1998.
- [17] A. Ferscha and G. Chiola, "Self-Adaptive Logical Processes: The Probabilistic Distributed Simulation Protocol," *Proc. 27th Ann. Simulation Symp.*, pp. 78–88. Los Alamitos, Calif.: IEEE CS Press, 1994.
- [18] A. Ferscha and J. Lüthi, "Estimating Rollback Overhead for Optimism Control in Time Warp," *Proc. 28th Ann. Simulation Symp.*, pp. 2–12. Los Alamitos, Calif.: IEEE CS Press, 1995.
- [19] A. Ferscha and M. Richter, "Massively Parallel Simulation of Business Process Models," A. Javor, A. Lehmann, and I. Molnar, eds., *Proc. 10th European Simulation Multiconf.*, pp. 377–381. SCS, 1996.
- [20] J. Ferscha, A. Johnson, and St. Turner, "Early Performance Prediction of Parallel Simulation Protocols," Y.M. Teo, W.C. Wong, T.I. Oren, and R. Rimane, eds., *Proc. First World Congress Systems Simulation, WCSS'97*, pp. 282–287, Singapore, IEEE CS Press, Sept. 1997.
- [21] R.M. Fujimoto, "Parallel Discrete Event Simulation," *Comm. ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.
- [22] D.O. Hammes and A. Tripathi, "Investigations in Adaptive Distributed Simulation," D.K. Arvind, R. Bagrodia, and J. Yi-Bing Lin, eds., *Proc. Eighth Workshop Parallel and Distributed Simulation (PADS'94)*, pp. 20–23, July 1994.
- [23] A.H. Jazwinski, *Stochastic Process and Filtering Theory*. New York: Academic Press, 1970.
- [24] Y-B. Lin and B.R. Preiss, "Optimal Memory Management for Time Warp Parallel Simulation," *ACM Trans. Modeling and Computer Simulation*, vol. 1, no. 4, pp. 283–307, Oct. 1991.
- [25] D. Nicol, "Automated Parallel Simulation of Timed Petri-Nets," *J. Parallel and Distributed Computing*, vol. 25, no. 1, pp. 60–74, Aug. 1995.
- [26] D.M. Nicol and S. Roy, "Parallel Simulation of Timed Petri-Nets. B. Nelson, D. Kelton, and G. Clark, eds., *Proc. Winter Simulation Conf.*, pp. 574–583, 1991.
- [27] H. Rajaei, R. Ayani, and L.E. Thorelli, "The Local Time Warp Approach to Parallel Simulation," R. Bagrodia and D. Jefferson, eds., *Proc. Seventh Workshop Parallel and Distributed Simulation*, pp. 119–126, Los Alamitos, Calif.: IEEE CS Press, 1993.
- [28] P.F. Reynolds, "A Spectrum of Options for Parallel Simulation," *Proc. Winter Simulation Conf.*, pp. 325–332, 1988.
- [29] S. Roy, "Massively Parallel SIMD Simulation of Discrete Time Stochastic Petri Nets," J.D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila, eds., *Proc. Winter Simulation Conf.*, pp. 229–236, 1994.
- [30] R.H. Shumway and D.S. Stoffer, "An Approach to Time Series Smoothing and Forecasting Using the EM Algorithm," *J. Time Series Analysis*, vol. 3, no. 4, pp. 253–264, 1982.
- [31] G.S. Thomas and J. Zahorjan, "Parallel Simulation of Performance Petri Nets: Extending the Domain of Parallel Simulation," *Proc. Winter Simulation Conf.*, 1991.



**Alois Ferscha** received his Mag degree in 1984, and a PhD degree in business informatics in 1990, both from the University of Vienna, Austria. In 1986, Dr. Ferscha joined the Department of Applied Computer Science at the University of Vienna, where he is presently an associate professor. He has published more than 50 technical papers on topics related to parallel and distributed computing, such as: computer-aided parallel software engineering, performance oriented distributed/parallel program development, parallel and distributed discrete event simulation, performance modeling/analysis of parallel systems and parallel visual programming. Currently, he is focused on distributed interactive simulation, WWW-based simulation, object oriented write-once run-anywhere software models, global net-centric computing, and multiuser distributed cooperative work environments. He has been the project leader for several national and international research projects such as: network computing, performance analysis of parallel systems and their workload, parallel visual programming, parallel simulation of very large office workflow models, distributed simulation on high performance parallel computer architectures, parallel simulation techniques and computer aided parallel software engineering. His current research project work is comprised of: modeling and analysis of time constrained and hierarchical systems (MATCH, HCM); broadband integrated satellite network traffic evaluation (BISANTE, ESPRIT IV); distributed cooperative environments (COOPERATE); and virtual enterprises. He has been a visiting researcher in the Dipartimento di Informatica at the Università di Torino, Italy; in the Dipartimento di Informatica at the Università di Genoa, Italy; in the Computer Science Department at the University of Maryland at College Park; and in the Department of Computer and Information Sciences at the University of Oregon, Eugene. He has served on the committees of several conferences such as PADS, SIGMETRICS, MASCOTS, TOOLS, PNPM, and ICS. In 1998, he served as the Program Committee chair for the IEEE/ACM/SCS 12th Workshop on Parallel and Distributed Simulation (PADS'98), and he is currently serving as the Program Committee chair for the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99).