Probabilistic Adaptive Direct Optimism Control in Time Warp

Alois Ferscha
Institut für Angewandte Informatik
Universität Wien
Lenaugasse 2/8, A-1080 Vienna, AUSTRIA
ferscha@ani.univie.ac.at

Abstract
In a distributed memory environment the communication overhead of Time Warp as induced by the rollback procedure due to "overoptimistic" progression of the simulation is the dominating performance factor. To limit optimism to an extent that can be justified from the inherent model parallelism, an optimism control mechanism is proposed, which by maintaining a history record of virtual time differences from the time stamps carried by arriving messages, and forecasting the timestamps of forthcoming messages, probabilistically delays the execution of scheduled events to avoid potential rollback and associated communication overhead (antimessages). After investigating statistical forecast methods which express only the central tendency of the arrival process, we demonstrate that arrival processes in the context of Time Warp simulations of timed Petri nets have certain predictable and consistent ARIMA characteristics, which encourage the use of sophisticated and recursive forecast procedures based on those models. Adaptiveness is achieved in two respects: the synchronization behavior of logical processes automatically adjusts to that point in the continuum between optimistically progressing and conservatively blocking, that is the most adequate for (i) the specific simulation model and (ii) the communication/computation speed characteristics of the underlying execution platform.
Keywords: Time Warp, Optimism Control, Forecast Models, Petri Nets, CM-5, RS6000 Cluster, PVM.

1 Introduction

The distributed simulation of event occurrences by a set of logical processes (LPs) executing asynchronously in parallel generates the same sequence of event occurrences that a sequential simulator would have produced, provided that every LP simulates events in nondecreasing timestamp order only. Although sufficient, it is not always necessary to obey this "local causality constraint" (*lcc*) [13] because events may be independent of each other with respect to their impact on the simulation future (concurrent events). Generally, therefore, a distributed discrete event simulation (DDES) insures correctness if the partial event ordering produced by the LPs executing concurrently is consistent with the total event ordering generated by a (hypothetical) sequential, discrete event simulation [17].

The Time Warp (TW) [16] DDES protocol, as opposed to the conservative Chandy-Misra-Bryant (CMB) protocols [20], optimistically ignores *lcc* by letting causality errors occur, but employs a *rollback* mechanism to recover from causality violations immediately upon or after their detection. The rollback procedure relies on the reconstructability of past states, which can be guaranteed by a systematic state saving policy and corresponding state reconstruction procedures. Performance inefficiencies caused by potentially excessive amounts of memory consumption for storing state histories, or by the waste of CPU cycles due to overoptimistically progressing simulations that eventually have to be "rolled back" are not present in CMB protocols. On the other hand, while CMB protocols need to verify whether it is *safe* to process an event (with respect to *lcc*), TW is not reliant on any information coming from the simulation model (e.g. lookahead). Furthermore, the severe performance degrade imposed on CMB by the mandatory deadlock management strategy is relieved from TW in a natural way, since deadlocks due to cyclic waiting conditions for messages able to make "unsafe" events safe to process by exploiting information from their timestamps can never occur. Another argument for the relaxation of *lcc* and TW is the hope for better model parallelism exploitation and an acceleration of the simulation over CMB since blocking is avoided.

Despite convincing advantages, TW is not devoid of shortcomings. The rollback mechanism is known to be prone to inefficient behavior in situations where event occurrences are highly dispersed in space and time. Such "imbalanced" event structures can yield recursive rollback invocations over 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*. In distributed memory multiprocessor environments or clusters of RISC workstations, i.e. environments where CPU

0-8186-7120-3/95 $04.00 © 1995 IEEE

performance is significantly tempered by the communication performance, rollback thrashing can cause excessively higher performance degrades (in absolute terms) as compared to shared memory or allcache systems [5]. Of outstanding practical interest for TW implementations in such environments is therefore the reduction of communication overhead induced by the protocol.

After presenting related work on optimism control for TW in Section 2, empirical observations from a distributed memory environment (CM-5) demonstrate that the communication behavior of TW is the dominating performance factor (Section 3). We show how the optimism in TW can be related to the parallelism available in the simulation model, which is extracted from observing message arrival patterns along the input channels of LPs. After discussing straightforward statistical methods for forecasting the next message's timestamp, a self-adaptive characterization procedure is worked out based on ARIMA (autoregressive-integrated moving average) stochastic processes to enable a *direct*, *probabilistic* and *self-adaptive* optimism control mechanism.

2 Background

Attempts to "limit the optimism" in TW in order to overcome rollback overhead potentials have appeared in the literature. Sokol, Briscoe and Wieland [26] propose to restrict optimistic simulation advancements to time *windows* that move over simulated time. In their *moving time window* (MTW) protocol, events e with an occurrence time $ot(e) > t + \Delta$ are not allowed to be simulated in the time window $[t, t + \Delta)$, but are postponed for the next time window $[t + \Delta, t + 2\Delta)$. Two events e and e' with $ot(e)$ and $ot(e')$ can therefore only be simulated in parallel if $\mid ot(e) - ot(e') \mid < \Delta$. Naturally, the protocol favors simulation models with a low variation of event occurrence distances relative to the window size. The implicit assumption that event occurrence times are distributed approximately uniformly in space, the obliviousness with respect to potentially "good" optimism beyond the upper window edge, as well as the difficulty to determine Δ such that enough events are admitted to make the simulation efficient have been the main criticisms of this approach.

Opposed to MTW, the *Breathing Time Bucket* (BTB) [27] employs adaptable "breathing" time cycles of variable widths (*time buckets*). Each time bucket contains the maximum number of causally independent events determined by the event horizon, i.e. the minimum occurrence time of any event scheduled in the previous bucket in some LP. "Risk-free" executions are attained by combining an optimistic windowing mechanism with a conservative message sendout policy, where the necessity of any antimessage is avoided by restricting potential rollback to affect only local history records (as in SRADS [8]). The Breathing Time Warp (BTW) [28] protocol combines features of MTW and BTB, based on the belief that the likelihood of an optimistically processed event being subject to a future correction increases with the distance of its timestamp from the global virtual time (GVT). Therefore, the sendout of event messages with timestamps 'distant' from GVT are delayed.

Other window-based optimism control mechanisms that appeared in the literature are the Bounded Time Warp (BTW) [30], which similar to MTW divides virtual time into equally sized intervals, but depletes all events from every interval before a new intervall is started, and MIMDIX [19], which probabilistically invokes resynchronization of LPs at regular time intervals to prevent LPs from excessive virtual time advancement. *Window-based throttling* [25] has also been used with the intent of preventing LPs from executing too far, but in addition, aggressive objects whose work has to be rolled back frequently are penalized with temporary suspension (*penalty-based throttling*). As such, the protocol described by Reiher and Jefferson is *adaptive* in the sense that it reacts in a selfcorrecting way to observed execution behavior.

The possibility of "adapting" the synchronization behavior of a DDES protocol to any desirable point within the spectrum between pure optimistic and pure conservative approaches has already been seen in [24]. Several contributions appeared along those ideas, one of the earliest being the *Adaptive TW concurrency control algorithm* (ATW) proposed by Ball and Hyot [3]. ATW temporarily suspends event processing if it has observed a certain number of *lcc* violations in the past, i.e. stop LVT advancement for a time period called the *blocking window* (BW). The size of BW is determined based on the minimum of a function describing wasted computation in terms of time spent in a (conservatively) blocked mode or a fault recovery mode as induced by the TW rollback mechanism. In [12], an optimal CPU delay interval is computed from an explicit cost model for the tradeoff between optimistically progressing and conservatively blocking the local simulation, established from a topological message arrival history map encoding the real-time – virtual-time increments (decrements) per message arrival as empirically observed during the simulation. The *probabilistic* DDES protocol [10] makes use of event causality probabilities to avoid communication overhead in TW by probabilistic throttling. Assuming that the occurrence of e in some LP$_i$ is *probabilistically causal* for a future event e' with $ot(e') = ot(e) + \delta$ in LP$_j$, i.e. e with $P[e \rightarrow e']$ changes the state variables read by e', then in cases where $P[e \rightarrow e'] < 1$, conservatively blocking until it is safe to process e' in LP$_j$ hinders producing potentially "good" simulation work. Clearly, in repeated executions of e, e' sequences with $P[e \rightarrow e'] \ll 1$, an optimistic strategy could have gained from a concurrent execution of e' and e most of the time. The protocol not only exploits locally (on a per channel basis in every LP) the probability of the forthcoming message being a straggler by taking into account the implicit probabilistic causalities, but also the architectural characteristics of the target platform like CPU speed and communication latencies. The *local adaptive protocol* (LAP) proposed by Hamnes and Tripathi [14], based on average LVT increments and average interarrival times

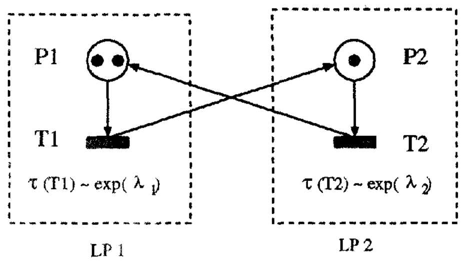

Figure 1: LP Simulation of a Stochastic Petri Net

(CPU time and simulated time) tries to estimate a real time blocking window. In order to prevent deadlocks, but also to break blocking conditions early, null messages are needed in LAP. According to Rajaei et. al. 's [23] classification of possibilities to regulate the degree of "aggressiveness" and "risk" in a DDES, LAP falls into the category *switching seamlessly between optimistic and conservative schemes*, whereas the previously described adaptive protocols are *limiting optimism* in TW.

Both ATW and the probabilistic protocol can be categorized as *direct optimism control* mechanisms, as opposed to *indirect optimism control*, where the individual LP's LVT progression is throttled via the availability of free memory. The *adaptive memory management* (AMM) scheme proposed by Das and Fujimoto [7] attempts a combination of controling optimism and an *automatic* adjustment of the amount of memory in order to optimize fossil collection, Cancelback [15] and rollback overheads. The Cancelback memory management scheme allows those memory spaces that are used for storing the most recent state and input-/output-history of some LP to be reclaimed selectively after TW has exhausted all available storage resources. Fossil collection relocates memory used for storing state information that will definitely not be reused by the rollback procedure due to GVT progression. It has been shown [1] that fossil collection in TW with Cancelback can always relocate *enough* memory for continuation of the simulation, given that a certain minimum amount of memory is physically available [18]. At this point, TW performance will be very poor due to frequent Cancelbacks. Increasing the amount of available memory will reduce the Cancelback frequency, such that absolute performance will have positive increments. But this at the same time will increase the rollback frequency, such that the rollback overhead will eventually start overwhelming the gain from reduced Cancelback overheads. AMM, by controling the amount of available memory, automatically adjusts to the "knee-point" of optimal TW performance.

3 Reducing Communication Overhead in TW

To demonstrate the potential gain of an adaptive direct optimism control mechanism for TW, we consider the Stochastic Petri Net (SPN) simulation model that has been used in [10]. The SPN (Figure 1) comprises two places (P1, P2) and two transitions (T1, T2) with exponentially distributed enabling delays $\tau(\mathrm{T1}) \sim exp(\lambda_1)$ and $\tau(\mathrm{T2}) \sim exp(\lambda_2)$. Together with infinite server (enabling) semantics, the SPN describes a continuous time, discrete event dynamic system with inherent model parallelism [9]. (The *occurrence time* $ot(\mathrm{T1}(\bullet_i))$ of T1 with the i-th token is determined by $t + X_i$ where $X_i \sim exp(\lambda_i)$ is an exponential variate with $\mathrm{pdf}_X = \lambda_i e^{-\lambda_i x}$. $ot(\mathrm{T1}(\bullet_i))$ does not depend on the presence or absence of any other token and T1 can "serve" multiple tokens simultaneously, thus expressing a notion of parallelism among individual tokens.) This example has been chosen since it is the smallest possible SPN structure able to express concurrency among event occurrences, where the degree of model parallelism can be scaled arbitrarily by simply adding tokens to the SPN, while at the same time arbitrary load imbalance can be imposed by mismatching timing parameters for T1 and T2.

In order to exploit this model parallelism in a distributed discrete event simulation, the SPN model is decomposed into two spatial regions which are assigned to two LPs (LP_1 and LP_2) as depicted in Figure 1. Two directed communication channels replacing the SPN arcs (T1, P2) and (T2, P1), thus interconnecting LP_1 and LP_2, are required to carry messages containing time stamped tokens $m = \langle k, \mathrm{P}, t \rangle$ that were generated by the firing of a transition. k is the number of tokens, P the destination place, and t a copy of the *local virtual time* (LVT) of the LP at the instant of that firing of the transition that produced the token. We call m a *tokenmessage*, since its purpose – much like an SPN arc – is to propagate tokens together with their timestamp from one spatial SPN region into another one that resides in a remote LP. In the sample SPN, the firing of a scheduled transition (internal event) always generates an external event, namely a message carrying a token. On the other hand, the receipt of an event message (external event) always causes a new internal event in the receiving LP, namely the scheduling of a new transition firing in the local event list EVL. Depositing tokens in a time consistent way into the target places requires the employment of a DDES synchronization protocol. Both, CMB and TW based protocols have been studied in the literature to synchronize the execution of spatially decomposed PNs [29, 2, 22, 6, 21].

3.1 TW Simulation of the SPN on the CM-5

We have implemented TW with the *lazy cancellation* rollback mechanism for the concurrent execution of PNs on the CM-5 using the CMMD message passing library.Executing the SPN simulation model in Figure 1 on the CM-5 empirically explains that communication is the major performance pitfall of TW implementations on distributed memory multiprocessors (Figure 2): The SPN with one token initially assigned to a place does not contain any model parallelism; the two LPs are blocked half of the time. With two tokens in the SPN we have very little model parallelism, and the LP simulation engines are overwhelmed with communication (when $\lambda_1 = 1$), the ratio of execution time used for processing events is less than 12%; the rest is wasted for communication, data structure manipulations and blocking due to the lack of events

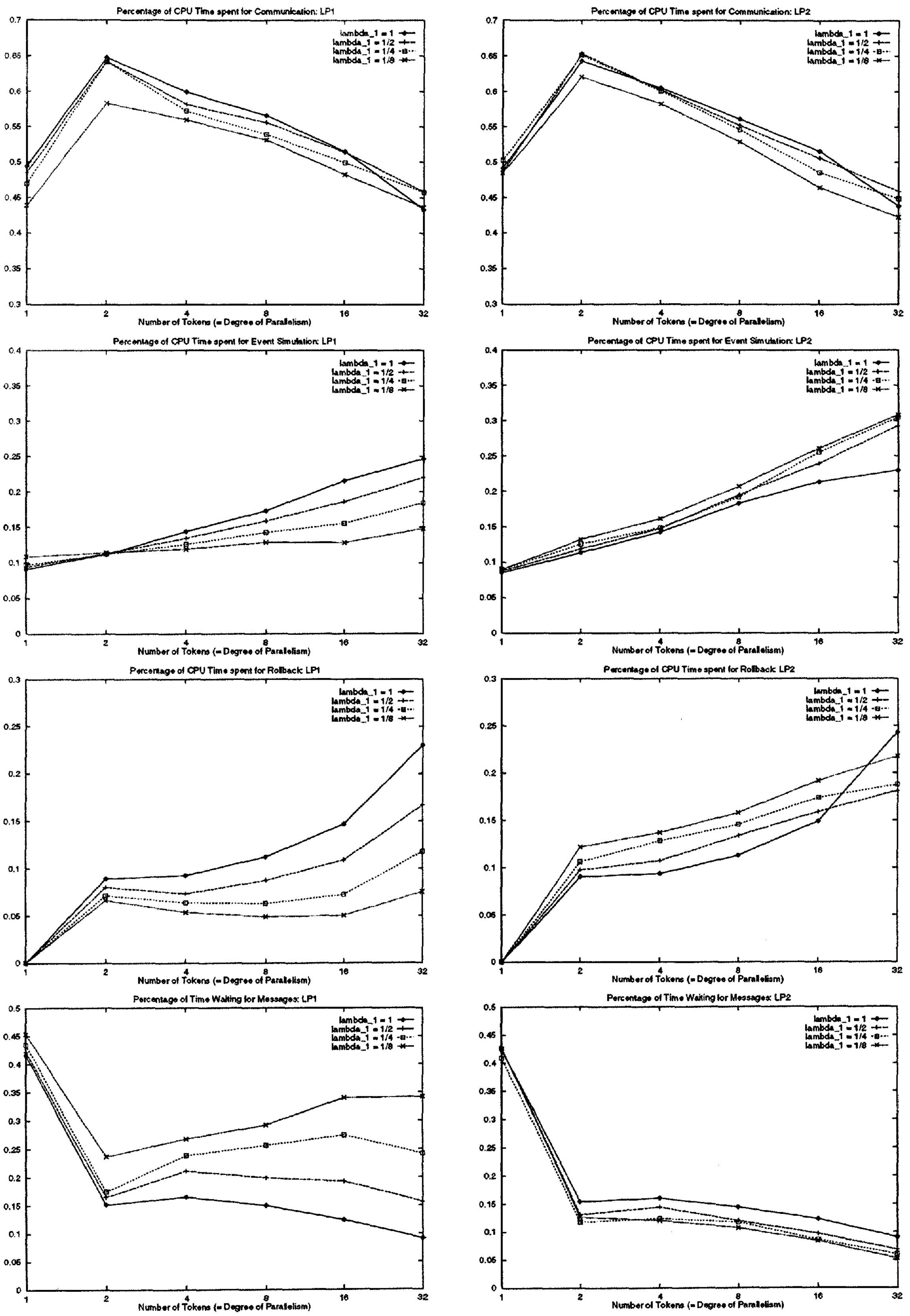

Figure 2: TW Performance (lazy cancellation) of LP_1 and LP_2 on CM-5

scheduled in the local EVL. The situation improves when more tokens are in the system: with a parallelism degree of 32 (32 tokens), about 25% of the CPU time can be used for executing internal events, but still the communication overhead is above 40%.

To investigate the impact of *lcc* violations due to inhomogeneous LVT increments in the communicating LPs on communication overhead induced by rollback we can (in our example) control the balance of LVT progress by the parameter λ_1: with $\lambda_1 = 1 (= \lambda_2)$ we have a balanced situation. Service at T1 takes on average as long as at T2 . Setting $\lambda_1 = 1/2$ makes T1 twice as fast (with respect to LVT progression) than T2, i.e. the enabling time is twice as long; the higher the expected enabling time of T1 (which is $\frac{1}{\lambda_1}$), the more tokens will reside in P1 (in steady state) enabling T1. The charts for rollback costs in Figure 2 empirically show that the smaller λ_1, the more rollbacks are induced in LP_2, imposing increasing rollback overhead on the CPU executing LP_2. Clearly, an LP with small LVT increments followed by an LP with high LVT increment will frequently force its successor to rollback, given they work at the same event processing speed. From the waiting time charts in Figure 2 it is observed, that a shift of load happens from LP_1 to LP_2 with increasing λ_1, giving LP_2 the chance to spend more CPU time on event execution. This is, unfortunately, at the expense of LP_1, which is forced to idle for load (tokens).

A consequence, in order to improve overall TW performance, is that rollback (and consequently communication-) overhead has to be avoided as far as possible. Reducing the absolute number of rollbacks/communications is the main issue of an optimism control mechanism in this context. Moreover, since the event structure of general simulation problems cannot be assumed to be stationary over the whole simulation interval, the capability of LPs to adapt to phases (where different degrees of optimism are advisable) emerging at runtime is demanded.

3.2 Gaining from Direct Optimism Control

The (synchronous) parallel execution of the sample SPN is illustrated in Table 1. In step 0, both LPs use precomputed random variates from their individual future lists and schedule events (EVL). (Let the future lists be FL(T1) = $(0.37, 0.17, 0.22, 0.34, \ldots)$ and FL(T2) = $(0.51, 0.39, 0.42, 0.05, \ldots)$ respectively.) In step 1, LP_1 and LP_2 execute their respective earliest internal events, generating external events (messages with copies of LVT) to be sent to the other LP, etc. At the beginning of step 3, LP_2 at LVT = 0.56 (= LVT at the end of step 2) faces the straggler (out-of-timestamp-order message) $\langle 1;P2;0.37 \rangle$ in its input queue IQ; the next element in LP_2's future list is 0.42. Since the effect of the straggler is in the local future of LP_2, i.e. $\langle T2@(0.37+0.42) \rangle$, the *lazy rollback* strategy applies and rollback is avoided at all. The event $\langle T2@0.79 \rangle$ is executed in that step, setting LVT = 0.79, and the outputmessage $\langle 1;P1;0.79 \rangle$ is generated (output queue, OQ) and sent at the end of the step. Unfortunately in step 4, a new straggler $\langle 1;P2;0.73 \rangle$ is observed in IQ of LP_2, but now with the effect that at time $t = 0.73 + 0.05 <$ LVT $= 0.79$ LP_2 is forced to roll back (Figure 1, top). Indeed, LP_2 in step 3 generated and sent out $\langle 1;P1;0.79 \rangle$ without considering any information whether the implicit optimism is justified or not. If LP_2 would have observed that it received "on the average" one input message per step, with an "average" timestamp increment of 0.185, it might have established a hypothesis that in step 4 a message is expected to arrive with an estimated timestamp of 0.37+0.185 = 0.555 (= timestamp of previous message + average increment). Taking this as an alarm for potential rollback, LP_2 could have avoided the propagation of the local optimistic simulation progression by e.g. delaying the sendout of $\langle 1;P1;0.79 \rangle$ for one step. This is illustrated in Figure 1, bottom: LP_2 just takes the input message from IQ and schedules the event $\langle T2@0.79 \rangle$ in EVL, but does *not* process it. Instead, the execution is delayed until the hypothesis upon the next message's timestamp is verified. The next message is $\langle 1;P1;0.73 \rangle$, the hypothesis can be dropped, and a new event $\langle T2@0.78 \rangle$ is scheduled and processed next. Apparently two rollbacks and the corresponding sending of antimessages could be avoided by applying a direct optimism control scheme, that employs *blocking* if there is empirical evidence (in the statistical sense) for a potential future rollback. In the next section we develop an adaptive optimism control mechanism, that by monitoring the arrival process of messages "on-the-fly" determines whether to let the simulation make full use of the available parallelism, or whether to throttle the optimism in order to prevent from costly rollbacks.

4 Probabilistic Direct Optimism Control

An indirect optimism control mechanism like AMM, although successful in shared memory environments, appears inappropriate for distributed memory systems since it potentially increases the number of rollbacks and thus the communication overhead. Instead, optimism control directly via throttling the simulation engine is advisable for distributed memory multiprocessors.

To be able to directly control the optimism in TW, each LP in our approach monitors the LVT progression on each of its incident channels, i.e. logs the timestamps of messages as they arrive. From the observed message arrival patterns, each LP formulates a hypothesis on the timestamp of the next message expected to arrive, and – related to statistical confidence in the forecast value – by means of throttling *adapts* to a synchronization behavior that is presumably the best tradeoff among blocking (CMB) and optimistically progressing (TW) with respect to this hypothesis in the current situation. Throttling is done *probabilistically* in the sense that *blocking* is induced with a certain probability.

Assume that the history over the last n message arrivals $A_k = (ts(m_{i-n+1}), ts(m_{i-n+2}), \ldots\ ts(m_i))$ is maintained in LP_l for every (input) channel $ch_{k,l}$, and that $\widehat{ts}(m_{i+1}))$ is an estimate for the timestamp of the forthcoming message m_{i+1}. Let the confidence $0 \leq \zeta(\widehat{ts}(m_{i+1}))) \leq 1$. express the "trust" in this estimate. Then LP_l having

	TW with unlimited optimism											
Step	LP1						LP2					
	IB	LVT	P1	EVL	OB	RB	IB	LVT	P2	EVL	OB	RB
0	—	0.00	2	T1@0.17; T1@0.37	—		—	0.00	1	T2@0.51	—	
1	—	0.17	1	T1@0.37	⟨ 1; P2; 0.17 ⟩		—	0.51	0	—	⟨ 1; P1; 0.51 ⟩	
2	⟨ 1; P1; 0.51 ⟩	0.37	1	T1@0.73	⟨ 1; P2; 0.37 ⟩		⟨ 1; P2; 0.17 ⟩	0.56	0	—	⟨ 1; P1; 0.56 ⟩	
3	⟨ 1; P1; 0.56 ⟩	0.73	1	T1@0.90	⟨ 1; P2; 0.73 ⟩		⟨ 1; P2; 0.37 ⟩	0.79	0	—	⟨ 1; P1; 0.79 ⟩	
4	⟨ 1; P1; 0.79 ⟩	0.90	1	T1@1.72	⟨ 1; P2; 0.90 ⟩		⟨ 1; P2; 0.73 ⟩	0.73	2	T2@0.78; T2@0.79	⟨ -1; P1; 0.79 ⟩	•
5	⟨ -1; P1; 0.79 ⟩	0.90	0	—	—	•	⟨ 1; P2; 0.90 ⟩	0.78	2	T2@0.79; T2@1.78	⟨ 1; P1; 0.78 ⟩	

	TW with "controlled" optimism											
Step	LP1						LP1					
	IB	LVT	P1	EVL	OB	RB	IB	LVT	P2	EVL	OB	RB
0	—	0.00	2	T1@0.17; T1@0.37	—		—	0.00	1	T2@0.51	—	
1	—	0.17	1	T1@0.37	⟨ 1; P2; 0.17 ⟩		—	0.51	0	—	⟨ 1; P1; 0.51 ⟩	
2	⟨ 1; P1; 0.51 ⟩	0.37	1	T1@0.73	⟨ 1; P2; 0.37 ⟩		⟨ 1; P2; 0.17 ⟩	0.56	0	—	⟨ 1; P1; 0.56 ⟩	
3	⟨ 1; P1; 0.56 ⟩	0.73	1	T1@0.90	⟨ 1; P2; 0.73 ⟩		⟨ 1; P2; 0.37 ⟩	0.56	1	T2@0.79	—	
4	—	0.90	0	—	⟨ 1; P2; 0.90 ⟩		⟨ 1; P2; 0.73 ⟩	0.78	1	T2@0.79	⟨ 1; P1; 0.78 ⟩	

Table 1: Reducing Communication Overhead with Probabilistic LP Simulation

```
program PADOC_Simulation_Engine( )
1      initialize();
2      while GVT ≤ endtime do
2.1        for all arriving messages $m$ do
               update(arrivalstatistics, $m$);
               if ts($m$) < LVT /* $m$ affects local past */
                 then /* rollback */
                      restore_earliest_state_before(ts($m$));
                      generate_and_sendout(antimessages);
                 else chronological_insert($m$, IQ);
               endif;
           od;
2.2        $\widehat{ts}$ = forecast(arrivalstatistics);
2.3        $\zeta(\widehat{ts})$ = confidence_in_forecast(arrivalstatistics);
2.4        if ts(first(EVL)) ≤ ts(first_nonnegative(IQ))
             then
                 if (1 − $P_\zeta$[exec first(EVL)]) ≤ rand()
                   then /* delay execution */
                        delay($\bar{s}$);
                   else process(first(EVL));
                 endif;
             else process(first_nonnegative(IQ));
           endif;
2.5        sendout(outputmessages);
2.6        fossil_collection(advance_GVT());
       od while;
```

Figure 3: PADOC Simulation Engine.

scheduled t_r as the transition to fire next, say at $ot(t_r)$, would execute the occurrence of t_r with some probability $P_\zeta[\text{execute } \langle t_r @ ot(t_r)\rangle]$, but would block for the average amount of CPU time $\bar{s}$ (used to simulate one transition firing) with probability $1 - P_\zeta$. The algorithm sketch of the PADOC (Probabilistic Direct Optimism Control) LP simulation engine in Figure 3 explains further details.

Note that in contrast to other adaptive TW mechanisms that compute an optimal delay window for blocking the simulation engine [3, 14, 12], the PADOC engine blocks for a fixed amount of real time (i.e. $\bar{s}$), but loops over the blocking decision, incrementally establishing longer blocking periods. By this discretization of the "blocking window" PADOC preserves the possibility to use information on the arrival process encoded in the timestamps of messages that arrive in between blocking phases. Algorithms based on variable size blocking windows fail to make use of intermediate message arrivals.

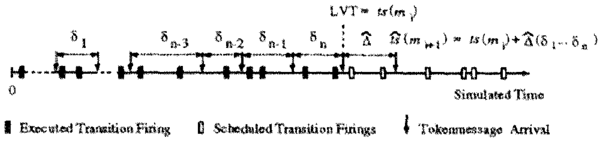

Figure 4: Message Timestamp Forecast

4.1 Incremental Forecast Methods

Predicting the timestamp of the forthcoming message m_{i+1} after having observed n arrivals is explained in Figure 4. Basically, by statistically analyzing the arrival instants $ts(m_{i-n+1}), ts(m_{i-n+2}), \ldots ts(m_i)$, an estimate $\widehat{ts}(m_{i+1}) = ts(m_i) + \hat{\Delta}(\delta_1, \delta_2, \ldots \delta_n)$ is generated, where $\delta_k = ts(m_{i-n+k}) - ts(m_{i-n+k-1})$ is the difference in timestamps of two consecutive messages. (Note that δ_k is negative if m_{i-n+k} is a straggler.)

The choice of the size of the observation history n as well as the selection of the forecast procedure is critical for the performance of the PADOC engine for two reasons: (*i*) the achievable prediction *accuracy* and (*ii*) the computational and space complexity of the forecast method. 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 distributed simulation protocol with an unacceptable amount of computational resource consumption. Therefore, *incremental* forecast methods of low memory complexity are recommended, i.e. procedures where $\widehat{ts}(m_{i+2})$ can be computed from the previous forecast $\widehat{ts}(m_{i+1})$ and the actual observation $ts(m_{i+1})$ in $\mathcal{O}(c)$ instead of $\mathcal{O}(cn)$ time.

Arithmetic Mean If no observation window is imposed on the arrival history, but all observed δ_j's are considered,

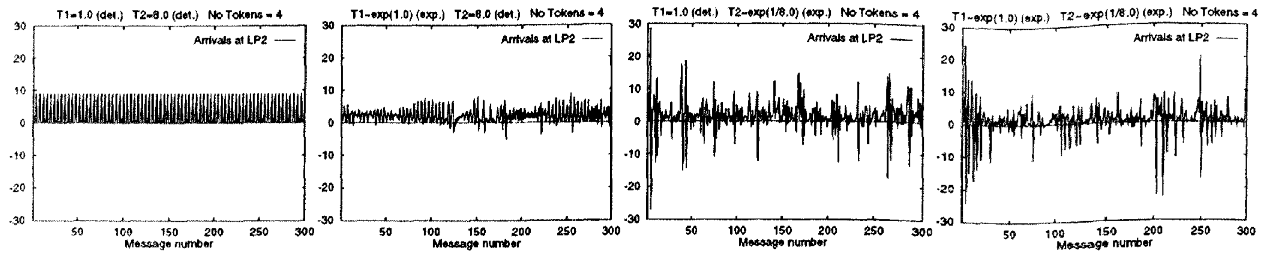

Figure 5: Arrival Processes as observed at LP$_2$ (CM-5)

then the observed mean $\widehat{\Delta}_i = \frac{1}{n}\sum_{j=1}^{n} \delta_j$ as an estimate of the timestamp of the forthcoming message has a recursive form. Upon the availability of the next time difference δ_{n+1}, $\widehat{ts}(m_{i+1})$ can be computed incrementally as:

$$\widehat{\Delta}_{n+1} = \frac{n\widehat{\Delta}_n + \delta_{n+1}}{n+1}.$$

Exponential Smoothing The arithmetic mean based forecast considers all observations δ_j as equally "important". A possibility to express the history as an exponentially weighted sum (e.g. give recent history higher "importance" than past history) is the exponential smoothing of the observation vector by a smoothing factor α ($|1-\alpha| < 1$). $\widehat{\Delta}$ in this case has the incremental form

$$\widehat{\Delta}_{n+1} = \alpha\delta_{n+1} + (1-\alpha)\widehat{\Delta}_n$$

$\alpha \approx 1$ gives a high weight to the last observation, and potentially yields a high variation in the forecasts. $\alpha \approx 0$ causes intense smoothing, making forecasts less reactive to shocks in the arrival process. We use the smoothing factor

$$\alpha = min_{\overline{\alpha}\in(0.1,0.2,\ldots,0.9)}\left(\sum_{i=1}^{n-1}[\delta_{i+1} - \widehat{\Delta}_{i+1}(\overline{\alpha})]^2\right),$$

which is periodically readjusted during the simulation.

Median Approximation The virtual time increments in general cannot be assumed to yield a nonskewed, unimodal distribution of values, as is implicitly assumed when the arithmetic mean is used as an index of central tendency. Particularly, if the frequency of time increments has a pdf skewed to the left, then the arithmetic mean is higher in value than the median, and would thus overestimate the next message's timestamp. A consequence would be "overpessimism" in the blocking policy. Forecast based on the median would use the estimate

$$\widehat{\Delta} = \begin{cases} \delta_{(\frac{n-1}{2})} & \text{n odd} \\ \frac{1}{2}(\delta_{(\frac{n}{2})} + \delta_{(\frac{n+1}{2})}) & \text{else} \end{cases}$$

which cannot be computed incrementally, as new timestamp increments have to be inserted in a sorted list of δ_i's to find the value of the median afterwards. As an approximation for the median we have developed the following supplement. Let $\mathcal{M} = \frac{\text{median}}{\text{mean}}$, which is a constant for every distribution (e.g. for the exponential distribution we have $(\frac{1}{\lambda}\ln 2)/\frac{1}{\lambda} = \ln 2$). Then with

$$\widehat{\mathcal{M}}(s) = \frac{\text{Median}(\delta_s + \delta_{s-1} + \ldots + \delta_{s-n+1})}{\frac{1}{n}(\delta_s + \delta_{s-1} + \ldots + \delta_{s-n+1})}$$

we find a forecast based on a median which is approximated by the arithmetic mean as

$$\widehat{\Delta} = \widehat{\mathcal{M}}\frac{1}{n}(\delta_1 + \ldots + \delta_n)$$

		Execution Time	%-Simulation		%-Rollback	
			LP1	LP2	LP1	LP2
DD	TW	0.46	12.7	13.7	5.8	16.1
	TW+M	0.60	12.5	33.4	12.1	5.7
	TW+S	0.65	16.4	35.9	11.5	6.3
	TW+A	0.62	23.7	41.0	6.8	5.4
SD	TW	0.68	10.9	13.8	9.8	17.2
	TW+M	0.84	19.2	35.0	6.8	8.9
	TW+S	0.96	19.4	37.7	6.8	8.4
	TW+A	0.64	19.8	41.8	5.8	6.8
DS	TW	0.64	10.7	13.8	8.4	17.9
	TW+M	1.51	22.1	35.6	8.2	8.7
	TW+S	0.66	17.1	39.2	6.0	8.2
	TW+A	1.17	24.3	43.5	5.8	9.5
SS	TW	0.91	11.0	15.0	11.7	15.3
	TW+M	0.57	16.6	37.6	7.3	6.9
	TW+S	1.03	19.8	37.9	7.4	9.6
	TW+A	0.91	23.1	41.6	6.8	9.3

Table 2: TW with M, S and A for CM-5.

The performance of the three "straightforward" forecast methods (arithmetic mean (M), exponential smoothing (S) and approximated median (A)) applied to the SPN in Figure 1 with 4 tokens in the initial marking and different timing scenarios is summarized in Table 4.1 (generated using the N-MAP virtual processor simulation tool with CM-5 performance settings [11]). In the scenario referred to as DD both T1 and T2 obey deterministic, but imbalanced timing (τ(T1) = 1, τ(T2) = 8). In the second case, SD, T1 has stochastic timing with $\tau(\text{T1}) \sim exp(1)$, but T2 is deterministically timed as τ(T2) = 8. Similarly, DS represents τ(T1) = 1 and $\tau(\text{T2}) \sim exp(1/8)$, whereas SS represents $\tau(\text{T1}) \sim exp(1)$, $\tau(\text{T2}) \sim exp(1/8)$. Note that in any case LVT progression in LP$_2$ is (on average) eight times higher than in LP$_1$ causing significant load imbalance and rollback (communication) overhead. (All forecast confidences are kept constant at $\zeta = 0.9$ for comparability, (TW) refers to TW with unlimited optimism.) Sample arrival process traces as collected on the CM-5 are depicted in Figure 5 for

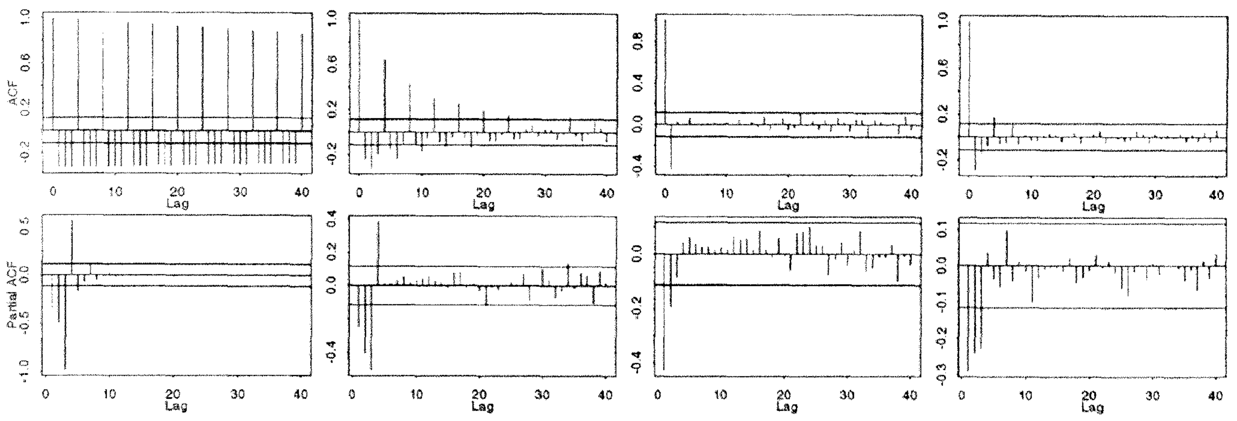

Figure 6: Autocorrelation and Partial Autocorrelation Function of Arrival Processes at LP$_2$ (CM-5)

the two LPs for DD (left), SD (half-left), DS (half-right) and SS (right).

Since the timestamp differences in DD toggle between $\delta = 0$ and $\delta = 9 = 8 + 1$, this deterministic behavior represents a neutral case for all methods, e.g. method M repeatedly overestimates and underestimates the next timestamp and thus cannot gain over TW in the long run. Overall execution time grows, however, since forecasting intrudes the simulation engine (stalls CPU cycles). The first quartuple of lines in Table 4.1 explains the order of intrusion induced by the various methods. In the case SD (second quartuple of lines in Table 4.1), method A finds the highest chances to avoid rollbacks and can outperform U. For DS, method M finds an absolute stress case, yielding a slowdown as compared to U. If the arrival process has two stochastic components (SS) both M and A can outperform U. The most important observation from Table 4.1 is, that all the methods are able to increase the percentage of overall execution time spent for simulating events over the communication overhead induced. Thus the optimism control schemes are even more promising for distributed memory environments, for which the communication/computation speed ratio is smaller than on the CM-5, e.g. a cluster of RISC workstations.

The main drawback of the forecast schemes M, S and A are that they cannot cope with transient "patterns" of arrivals, but do respect only a central tendency of timestamp increments. Arrival patterns that show certain regularities, or at least some correlation in the time increments, can yield to stress cases as was seen above. Therefore forecast methods able to identify correlations and to predict next events at the maximum likelihood of those correlations are demanded.

4.2 ARIMA Forecasts

In this section we follow the idea of considering the arrival process as an unknown stochastic processes $\{X_t\} = (X_1, X_2, \dots X_n)$, where $X_1, X_2, \dots X_n$ are a series of in-

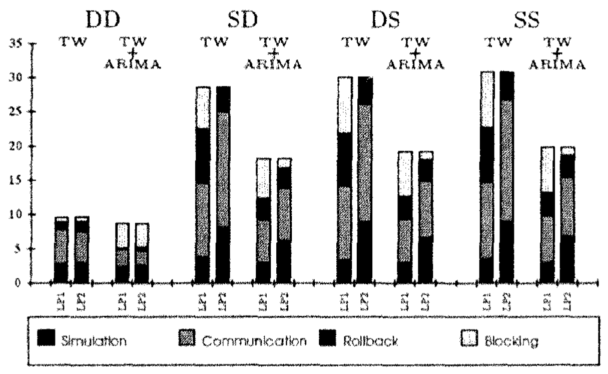

Figure 8: TW with ARIMA for RS6000 Cluster

stances of a random variable. Specifically $X_t = \delta_t - \bar{\delta}$ are the empirically observed timestamp differences, transformed by the series mean $\bar{\delta}$. If $\{X_t\}$ are statistically dependent variables, the arrival process can be modeled by an integrated autoregressive moving average process ARIMA$[p, d, q]$ (see e.g. [4])

$$\phi(B)\nabla^d X_t = \theta(B)\epsilon_t \qquad (1)$$

where $\nabla^d = (1 - B)^d$ is the d-fold differencing operator, and B the backward shift operator defined by $B^i X_t = X_{t-i}$ $i = 0, \pm 1, \pm 2, \dots$. This means that for e.g. $d = 2$, the process $Y_t = \nabla^2 X_t = X_t - 2X_{t-1} + X_{t-2}$ is assumed to be a *stationary* ARMA$[p, q]$ (=ARIMA$[p, 0, q]$), composed by a pure autoregressive process of order p (AR$[p]$) explaining Y_t as a dependency $Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \epsilon_t$, ϵ_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 $E(\epsilon_i) = 0$, $Var(\epsilon_i) = \sigma_\epsilon^2$ and $E(Y_t) = 0$.

Looking at the arrival process as obtained on the CM-5 for LP$_2$ of the SPN in Figure 1 (Figure 5) and the corresponding autocorrelation (ACF) and partial ACF (Fig-

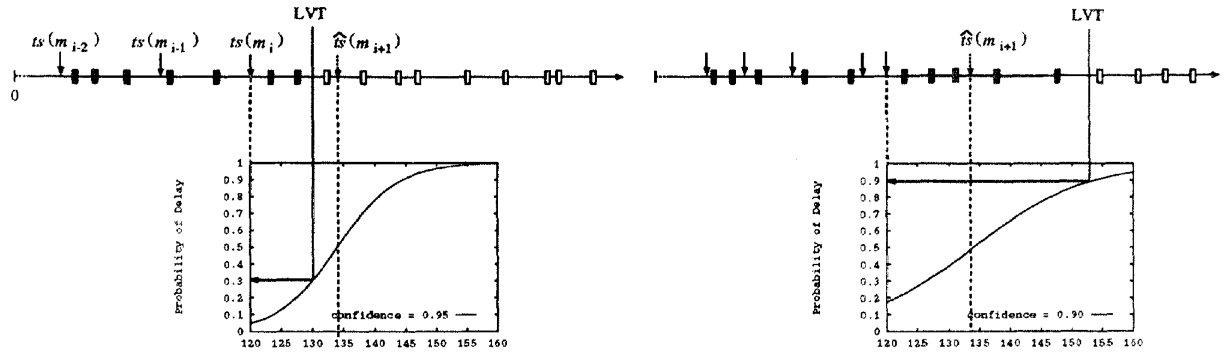

Figure 7: Probabilistic Direct Optimism Control with ARIMA Forecasting

ure 6), we find high positive correlation after every fourth lag for DD (Figure 6, left) obviously due to the four tokens in the SPN. The case SD (Figure 6, half-left) leads us to hypothesize that the arrival process is AR, since we find ACF dying down in an oscillating damped exponential fashion. This is also intuitive because the deterministic component in the process ($\tau(T2) = 8$) dominates the stochastic one ($\tau(T1) \sim exp(1)$). DS (Figure 6, half-right) gives evidence for a suitable representation of the the arrivals as a MA process, becuase ACF has a single spike at lag 1, and PACF dies down, etc.

For the automated characterization of the arrival process as an ARIMA$[p,d,q]$ process, the classical Box-Jenkins procedure can be adapted:

1. Model Order Identification First, the order of the ARIMA model, $[p,d,q]$ is identified. Since the theoretical partial autocorrelations $\varrho_{k-1}(k)$ to the lag k vanish after some $k > p$ for a pure AR$[p]$, the order of such a process can be approximated from the empirical partial autocorrelations $r_{k-1}(k)$. Similarly, for a pure MA$[q]$, the theoretical autocorrelations $\varrho(k)$ vanish after some $k > q$, such that again the empirical data (autocorrelations $r(k)$) can be used to approximate the order. For a combined ARMA$[p,q]$ process, the Akaike-criterion, i.e. the combination (p,q) that minimizes $AIC(p,q) = \log \widehat{\sigma}^2_{p,q} + \frac{2}{n}(p+q)$ approximates the order. ($\widehat{\sigma}^2_{p,q}$ is a Maximum-Likelihood estimate of the variances σ^2_ϵ of the underlying white noise error.) The Akaike-criterion has a more general form for ARIMA$[p,d,q]$ processes. Indeed, as intuitively recognized, we find best order fittings e.g. for SD as ARIMA$[3,0,0]$ or for DS as ARIMA$[0,0,4]$.

2. Model Parameter Estimation In the next step, the parameters in (1) ($\phi_1,\ldots,\phi_p$ and $\theta_1,\ldots,\theta_q$) are determined as maximum likelihood estimates from the empirical data, i.e. the estimates $\hat{\phi}_1,\ldots,\hat{\phi}_p$ and $\hat{\theta}_1,\ldots,\hat{\theta}_q$ that minimize the square sum $S^2(\hat{\phi}_1,\ldots,\hat{\phi}_p,\hat{\theta}_1,\ldots,\hat{\theta}_q) = \sum_{t=-\infty}^{n} \tilde{\epsilon}_t^2$ of the residuals $\tilde{\epsilon}_t = \delta_t - \hat{\phi}_1\delta_{t-1} - \ldots - \hat{\phi}_p\delta_{t-p} - \hat{\theta}_1\tilde{\epsilon}_{t-1} - \ldots - \hat{\theta}_q\tilde{\epsilon}_{t-q}$ are used as the model parameters.

3. Model Diagnostics/Verification A well know method to validate the model with the estimates $\hat{\phi}_1,\ldots,\hat{\phi}_p$ and $\hat{\theta}_1,\ldots,\hat{\theta}_q$ is the Portmonteau lack-of-fit-test, which tests whether the residuals $\tilde{\epsilon}_t$ are realizations of a white noise process. The confidence level $(1-\alpha)$ of the test can be used as a measure to quantify the "trust" in the model and, as a consequence, in the forecast.

4. Forecast Finally, the (recursive) Durbin-Levinson method provides an algorithm for the one-step (or k-step) best linear prediction for $\widehat{X}_{t+1}$.

At a confidence level $\zeta = (1-\alpha)$, the PADOC simulation engine (Figure 3, in Step 2.4) executes the next scheduled a transition firing with probability

$$P_\zeta[\text{exec first(EVL)}] = 1 - \left(1 + e^{-\left(\frac{LVT - \widehat{ts}}{\zeta(1-\zeta)100}\right)}\right)^{-1}, \quad (2)$$

otherwise the CPU is blocked for $\bar{s}$ time units. Figure 7 explains the blocking probability (2) related to the confidence level $\zeta = (1-\alpha)$: The higher the confidence ζ, the steeper the ascent of the delay probability as LVT progresses towards $\widehat{ts}$. (Steepness of the sigmoid function in Figure 7 (left) with $\zeta = 0.95$ is higher than in Figure 7 (right) $\zeta = 0.90$). Note also that after LVT progression in LP$_j$ has surpassed the estimate $\widehat{ts}$ (Figure 7, right), delays become more and more probable, expressing the increasing rollback hazard the LP runs into. A general observation is that with $\zeta \approx 1$, PADOC imposes a synchronization behavior close to CMB, whereas with $\zeta \approx 0$, optimism is as unlimited as in (plain) TW. Moreover, by periodically rebuilding the ARIMA model, the PADOC scheme adapts the LP to a synchronization behavior directly reflecting the inherent model parallelism, and also copes with *transient* arrival processes.

Clearly, the ARIMA approach for optimism control is much more expensive in space and execution time than the previous methods M, S and A. Since the implementation of steps 1. - 3. of the Box Jenkins procedure is still under way, we have provided the simulator with an ARIMA model computed off-line for the performance comparison reported in Figure 8. For the SPN in Figure 1 with a model parallelism of 100 (i.e. 50 tokens in P1 and P2) and an N-MAP execution with RS6000 and PVM 3.2 performance characteristics, we find the ARIMA based method able to (significantly) outperform TW (and all other approaches not shown), while being at least as good as those in stress cases like DD. The same scenario executed for the CM-5 revealed about the same performance characteristics for the ARIMA method, whereas M, S, and A gained less. M and S tend to more consistent forecasts and therefore better performance as model parallelism increases, whereas ARIMA is not significantly sensitive to model parallelism.

5 Conclusion

A probabilistic direct optimism control (PADOC) mechanism for the TW distributed discrete event simulation protocol has been presented. Our simulation engine, by temporarily blocking the processing of internal events, avoids the generation and sendout of messages in states for which it is likely that they will have to be "rolled back". Vice versa, every LP tends to await messages that influence the local causality among events with high probability, in order to avoid causality violations. A statistical analysis of the message arrival history is used to make forecasts for the timestamps of future messages, thus enabling every LP to adapt its local synchronization behavior to the most efficient strategy with respect to the anticipated future. Two classes of forecast methods are studied: (*i*) for estimates based on (weighted) means, efficient (incremental) procedures can be implemented causing negligible or minor intrusion on the simulation engine. Those methods (arithmetic mean, exponential smoothing and median approximation) however cannot cope well with seasonal, nonstationary arrival process, and are thus prone to pathological behavior. (*ii*) at the cost of higher computational complexity, more sophisticated forecast methods with a much higher prediction precision in the case of periodic or seasonal (correlated) channel (virtual) time increments can be used. Specifically, the time increment process can be modelled as an integrated autoregressive moving average process (ARIMA$[p,d,q]$), and the probabilities for delaying the execution of the next internal event can be directly related to the confidence in the model approximation.

The PADOC mechanism gains adaptiveness in the sense that, independent of the ratio of the communication and computation speed of the target platform, the synchronisation policy is adjusted automatically to that point in the continuum between TW and CMB protocols, that is most appropriate for the parallelism inherent in the simulation model. Forecasting based on ARIMA$[p,d,q]$ models, moreover, makes the simulation engine also able to adapt to transient (nonstationary) arrival processes.

Acknowledgements This paper was elaborated while the author was visiting the University of Maryland, supported by a grant from the Academic Senate of the University of Vienna. The use of the resources at the Computer Science Dept. and the CM-5 at UMIACS are gratefully acknowledged. The work was partially supported by the Austrian Federal Ministry of Science and Research under grant CEI GZ 308.926 and the Oesterreichische Nationalbank under grant No. 5069. The author wishes to thank the anonymous referees for valuable comments on a preliminary version of this paper.

References

[1] I. F. Akyildis, L. Chen, R. Das, R. M. Fujimoto, and R. F. Serfozo. The Effect of Memory Capacity on Time Warp Performance. *Journal of Parallel and Distributed Computing*, 18(4):411–422, August 1993.

[2] H. H. Ammar and S. Deng. Time Warp Simulation of Stochastic Petri Nets. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pages 186 – 195. IEEE-CS Press, 1991.

[3] D. Ball and S. Hoyt. The Adaptive Time-Warp Concurrency Control Algorithm. *Distributed Simulation. Proc. of the SCS Multiconference on Distributed Simulation*, pages 174 – 177, 1990. Simulation Series, Vol. 22, No. 1.

[4] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer Verlag, New York, 1991.

[5] Ch. D. Carothers, R. M. Fujimoto, and P. England. Effect of Communication Overheads on Time Warp Performance: An Experimental Study. *Proc. of the 8th Workshop on Parallel and Distributed Simulation*, pages 118–125, 1994.

[6] G. Chiola and A. Ferscha. Distributed Simulation of Petri Nets. *IEEE Parallel and Distributed Technology*, 1(3):33 – 50, August 1993.

[7] S. R. Das and R. M. Fujimoto. An Adaptive Memory Management Protocol for Time Warp Parallel Simulation. *Proc. of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 201–210, 1994.

[8] Ph. M. Dickens and P. F. Reynolds. SRADS with Local Rollback. *Proc. of the SCS Multiconference on Distributed Simulation Vol. 22 (1)*, pages 161–164, 1990.

[9] A. Ferscha. Concurrent Execution of Timed Petri Nets. *Proc. of the 1994 Winter Simulation Conference*, pages 229 – 236, 1994.

[10] A. Ferscha and G. Chiola. Self Adaptive Logical Processes: The Probabilistic Distributed Simulation Protocol. *Proc. of the 27th Annual Simulation Symposium*, pages 78–88, 1994.

[11] A. Ferscha and J. Johnson. Performance Oriented Development of SPMD Programs Based on Task Structure Specifications. *Parallel Processing: CONPAR94-VAPP VI*, LNCS 854, pages 51–65. Springer Verlag, 1994.

[12] A. Ferscha and J. Lüthi. Estimating Rollback Overhead for Optimism Control in Time Warp. *Proc. of the 28th Annual Simulation Symposium*, 1995. to appear.

[13] R. M. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[14] Donald O. Hamnes and Anand Tripathi. Investigations in Adaptive Distributed Simulation. *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, pages 20–23, 1994.

[15] D. Jefferson. Virtual Time II: The Cancelback Protocol for Storage Management in Time Warp. *Proc. of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 75–90, 1990.

[16] D. Jefferson and H. Sowizral. Fast Concurrent Simulation Using the Time Warp Mechanism. *Distributed Simulation 1985*, pages 63–69, 1985.

[17] D. A. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[18] Yi-Bing Lin and Bruno R. Preiss. Optimal Memory Management for Time Warp Parallel Simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(4):283–307, October 1991.

[19] V. Madisetti, D. Hardaker, and R. Fujimoto. The MIMDIX Operating System for Parallel Simulation. *Proc. of the 6th Workshop on Parallel and Distributed Simulation*, pages 65 – 74, 1992.

[20] J. Misra. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, 18(1):39–65, 1986.

[21] D. Nicol and W. Mao. Automated Parallelization of Timed Petri-Net Simulations. submitted for publication, 1994.

[22] D. M. Nicol and S. Roy. Parallel Simulation of Timed Petri-Nets. *Proc. of the 1991 Winter Simulation Conference*, pages 574 – 583, 1991.

[23] H. Rajaei, R. Ayani, and L. E. Thorelli. The Local Time Warp Approach to Parallel Simulation. *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, pages 119–126, 1993.

[24] P. F. Reynolds. A Spectrum of Options for Parallel Simulation. *Proc. of the 1988 Winter Simulation Conference*, pages 325 – 332, 1988.

[25] P. Reiher and D. Jefferson. Limitation of Optimism in the Time Warp Operating System. *Proc. of the 1989 Winter Simulation Conference*, pages 765 – 769, 1989.

[26] L. M. Sokol, D. P. Briscoe, and A. P. Wieland. MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution. *Proc. of the SCS Multiconf. on Distributed Simulation*, pages 34 – 42, 1988.

[27] J. Steinman. SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation. *International Journal in Computer Simulation*, 2:251 – 286, 1992.

[28] J. S. Steinmann. Breathing Time Warp. *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, pages 109–118, 1993.

[29] G. S. Thomas. Parallel Simulation of Petri Nets. Technical Report TR 91-05-05, Dep. of Computer Science, University of Washington, May 1991.

[30] St. Turner and M. Xu. Performance Evaluation of the Bounded Time Warp Algorithm. *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, pages 117–126, 1992.