Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

AGGRESSIVENESS/RISK EFFECTS BASED SCHEDULING IN TIME WARP

Vittorio Cortellessa

Computer Science and Electrical Engineering
CEMR, West Virginia University
Morgantown, WV 26506-6109, U.S.A.

Francesco Quaglia

Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, 00198 Roma, ITALY

ABSTRACT

The Time Warp synchronization protocol for parallel discrete event simulation is characterized by aggressiveness and risk. The former property refers to greediness in the execution of unsafe events. The latter one refers to greediness in the notification of new events produced by aggressive event execution. Both these properties are potential sources for rollback occurrence/spreading. In this paper we present a scheduling algorithm for the selection of the next LP to be run on a processor which tends to keep low the joint impact of these two properties on the experienced amount of rollback. Reduction of negative effects of aggressiveness and risk is achieved by giving higher priority to the LPs whose next event has low probability to be undone due to rollback and has low fan-out that is, notifies few new events. Our algorithm differs from most previous solutions in that they miss a direct control on the effects due to risk. These solutions could originate poor performance for applications with high variance of the number of new events notified which is an indicator of the risk associated with event execution.

1 INTRODUCTION

“Optimism” underlying the Time Warp synchronization protocol for parallel discrete event simulation (Jefferson 1985) has been defined, see (Reynolds 1988), as the union of two distinct properties: *aggressiveness* and *risk*. Aggressiveness is the property by which the logical processes (LPs) greedily execute simulation events without taking into account the event safety. Risk is the property by which the LPs greedily notify new events produced by aggressive (unsafe) event execution. Both these properties are potential sources for causality errors, therefore they directly determine the amount of rollback, and thus the final performance perceived.

A primary way to bound the effects of aggressiveness and risk on rollback consists of a direct control on optimism. This includes throttling (Ferscha 1995, Ferscha and Luthi 1995, Srinivasan and Reynolds 1998), time windows (Lubachevsky, Weiss and Shwartz 1989, Sokol, Briscoe and Wieland 1988, Steinman 1993, Turner and Xu 1992), risk free synchronization (Dickens and Reynolds 1990) and others (Ball and Hoyt 1990, Madisetti, Hardaker and Fujimoto 1993). Interested readers can refer to (Srinivasan and Reynolds 1998) for a recent classification of control strategies. However, for simulations of large/complex systems it is extremely likely that multiple LPs are hosted by the same processor, therefore a form of control on the effects of aggressiveness/risk should be implemented also at the level of the scheduling algorithm for the selection of the next LP to be run on a processor.

Controlling the effects of aggressiveness at the level of the scheduling algorithm means scheduling with higher priority the LPs whose next event is recognized to have, based on some criterion, lower probability to be eventually rolled back. Controlling the effects of risk means taking a scheduling decision as a function of the number of new events to be notified, namely the fan-out, and also of their timestamps. Specifically, an event producing new events with lower timestamps should be executed promptly; otherwise, upon the delivery of the new events their timestamps get more likely to be lower than the local virtual time (LVT) of the recipient LPs, thus causing *primary* rollbacks. On the other hand, the execution of an event producing a large number of new events that is, with large fan-out, should be thwart by the scheduling algorithm; otherwise, in case it is undone, there is high likelihood of *secondary* rollbacks due to a large number of antimessages.

In this paper we introduce an adaptive scheduling algorithm, namely Aggressiveness/Risk Effects based Scheduling (ARES), aimed at keeping low the joint impact of aggressiveness and risk on the amount of rollback. ARES first selects a set of candidate LPs to be run, in the way that their next event shows low probability to be eventually undone; then it chooses an LP among the candidates in the way to minimize the number of new events to be notified. Adaptiveness relies in that the first step of the scheduling decision is performed using performance feedback from the simulation.

ARES is designed as a modification of the classical Lowest-Timestamp-First algorithm (LTF) with aim at reducing the effects of risk. Adaptiveness guarantees that ARES shows performance which is at least equal to that of LTF. The know-how to apply ARES consists of knowledge of the fan-out of each event type. In most simulations this can be obtained off-line by the event code structure. Nevertheless, for the case of code branching or loops having an impact on the fan-out, a light run-time prediction mechanism could be coupled with ARES.

Timestamps of events that have not yet been produced (which are actually a risk index) are not considered in the scheduling decision performed by ARES. In other words, risk associated with an event execution is quantified only as a function of the fan-out of the event. This design choice derives from two reasons. First, the issue of reducing the effects of risk on primary rollbacks as a function of timestamps of future events has been already addressed in (Ronngren and Ayani 1994). Second, reducing the effects of risk on rollback giving higher priority to low fan-out event is likely to produce an additional performance improvement due to a possible reduction of the real communication cost. Specifically, undoing an event with low fan-out implies low waste of communication time because few notification messages must be revoked, and, furthermore, we get low communication overhead due to antimessages.

Target applications for ARES are simulations that could heavily suffer from risk effects, such as simulations of network protocols with broadcast/multi-cast queries where the number of new events notified by an event execution could assume highly different values depending on the type of event executed. This class of protocols encompasses some protocols for Web document retrieve among peer proxies (Fan et al. 1998, Wessel and Claffy 1998).

The performance improvements achievable using ARES are quantified through a performance study of a classical synthetic benchmark on a cluster of PCs connected by a high speed Myrinet switch.

The remainder of the paper is organized as follows. Section 2 presents an overview of existing scheduling algorithms. In Section 3 the ARES algorithm is introduced. Performance data are reported in Section 4.

2 RELATED WORK

The standard solution for the scheduling problem is the Lowest-Timestamp-First algorithm (LTF), which always schedules the LP having as next event to be executed the one with the minimum timestamp (Lin and Lazowska 1991). LTF implicitly assumes that the event with the minimum timestamp has the lowest probability to be rolled back in the future of the simulation execution as it is the closest one to the Global-Virtual-Time (GVT) that is, the commitment horizon of the simulation.

Another scheduling algorithm, namely Lowest-Local-Virtual-Time-First (LLVTF), gives higher priority to LPs having lower LVTs (Preiss, Loucks and MacIntyre 1994). In particular, LLVTF chooses for the execution the non-executed event of the LP with the lowest LVT value. As the LVT of the LP moves up to the event timestamp upon the execution, the objective of this scheduling algorithm is to reduce the probability for any LP to remain back in simulated time.

In (Palaniswamy and Wilsey 1994) an Adaptive Control based scheduling algorithm (AC) has been presented. In this solution, statistics on the past behavior of an LP are collected to establish the “useful work” of the LP (computed as the frequency of committed events of the LP); higher priority is assigned to the LPs having higher values of their useful work.

A rather different solution, namely Service Oriented scheduling (SO), is presented in (Ronngren and Ayani 1994). The idea behind this solution is to try to produce and deliver as soon as possible events not yet produced that will have low timestamps. This is done in order to promptly deliver these events to the recipient LPs, thus reducing the probability of timestamp order violations. Such an approach needs the capability of the LPs to predict the timestamps of events that have not yet been produced. SO gives the highest scheduling priority to the LP whose next event will produce the event with the minimum predicted timestamp.

A Probabilistic scheduling algorithm (P) has been presented in (Som and Sargent 1998). The consideration at the basis of this algorithm is that low amount of rollback can be obtained if the event selected for execution is the one with the minimum *real* probability to be rolled back in the future; this event may be different from that with the minimum timestamp. In this solution statistics on the past behavior of the LPs are maintained in order to estimate the probability for the next event of any LP to be not rolled back in the future. The event of the LP associated with the highest estimated probability value is selected for the execution.

In (Quaglia 2000) a State Based scheduling algorithm (SB) has been presented. In this algorithm, the scheduling priority of any LP is computed using state information related to the LPs in its immediate predecessor set. Specifically, higher priority is assigned to the LPs, if any, whose next event could be rolled back only conditional a rollback occurs on an LP in their immediate predecessor sets. If no such an LP is detected at the scheduling time, then SB acts as the classical LTF.

Finally, in (Quaglia and Cortellessa 2000) a Grain Sensitive scheduling algorithm (GS) has been presented. This algorithm gives higher priority to the LPs having non-executed events with low timestamp values and small expected granularity. This solution tends to delay the execution of large grain events that, if rolled back, could produce large waste of CPU time. Delaying the execution of these events makes lower their probabilities to be eventually undone.

Most of these algorithms, except SO, do not take into account directly the effects of risk in the scheduling decision. SO takes these effects into account at some extent because the scheduling decision relies on predicted timestamps of new events that have not yet been notified. Differently from SO, ARES quantifies the effects of risk as a function of the event fan-out, instead of timestamp values. Therefore, ARES is expected to provide better performance for the case of simulations with high fan-out variance for different event types. As pointed out before, simulations of protocols for data retrieve on the Web could exhibit this feature.

3 AGGRESSIVENESS AND RISK EFFECTS BASED SCHEDULING

In this section the ARES algorithm is described. We first discuss the mathematical background for the algorithm, then we describe the algorithm structure. Finally we report some considerations on long-term effects of ARES.

3.1 A Tradeoff Between Aggressiveness and Risk Effects

In Figure 1 the event queues of four LPs hosted by the same processor are shown, restricted to non-executed events. Labeled circles represent events and arrows exiting circles represent new events that will be produced by the event execution. As an example, event a of LP1 produces three new events, while event f of LP4 produces only one new event.

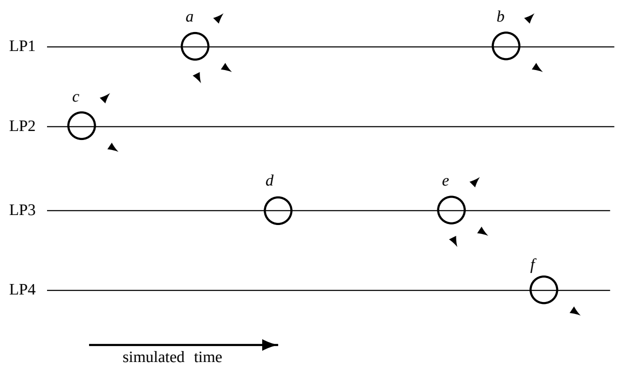

Figure 1: Event Queues of the LPs and Future Notifications

Out of any scheduling policy, but satisfying causality constraints (i.e. timestamp ordering at each LP), the set of events candidate to be executed is $S = \{a, c, d, f\}$. The classical LTF algorithm would schedule LP2 for running, because c is the event with minimum timestamp and, in the common belief, it should have the lowest probability to be eventually undone. This choice has the direct objective to bound the effects of aggressiveness on rollback.

Associated with each event $x \in S$ there is also risk, whose effects on rollback are a function of the number of new events produced, namely the fan-out, and of their timestamps. As stated in the Introduction, we limit our approach by quantifying the effects of risk only as a function of the fan-out. This solution aims at bounding the spreading of secondary rollbacks due to antimessages and, additionally, shows the potential for reducing the real communication cost.

For each event in S we can define the following simple model for the effects due to risk. Let us denote with $P(x)$ the probability for an event $x \in S$ to be eventually undone if selected for the execution and with $fo(x)$ the fan-out of the event x. Then the model can be expressed as:

$$C(x) = P(x) \cdot fo(x) \tag{1}$$

In other words, $C(x)$ measures the expected number of antimessages associated with the undoing of event x. While building the model we have implicitly assumed that the strategy for sending antimessages is aggressive that is, antimessages are sent as soon as the LP rolls back and undoes x. Correction factors could be introduced to cope with lazy strategies. According to our perspective, in which timestamps are discarded in the evaluation of risk effects, $C(x)$ is actually a measure for the effects of risk.

Coming back to the example in Figure 1 we get that the event d has risk effects $C(d) = 0$, although it could contribute to rollback due to aggressiveness effects (this effect can be quantified as $P(d)$). Overall, according to the common belief, c is the event whose aggressiveness effects should be minimal, as it has the lowest timestamp; on the other hand, as stated by the model in (1), d is the event whose risk effects should be minimal. As a consequence, LP2 represents the best scheduling choice with respect to aggressiveness effects (as its next event is supposed to have the minimum probability to be eventually undone), while LP3 represents the best scheduling choice in terms of risk effects (as its next event does not notify any new event). In the next section we show how to manage this tradeoff in practice.

3.2 Managing the Aggressiveness/Risk Effects Tradeoff

In order to manage the tradeoff of previous section, we exploit some ideas presented in the GS scheduling algorithm in (Quaglia and Cortellessa 2000). Specifically, in that paper a solution for constructing flexible scheduling decisions as a function of the granularity of the events has been presented. This solution relies on a notion of Scheduling Interval (SI) associated with a so-called Scheduling Window (SW).

Specifically, denoting with min_ts the minimum timestamp value among all the events in the set S (recall that this set contains the next events of the LPs hosted by the same processor), the authors noted that events belonging to S and having timestamp in narrow time proximity to min_ts are likely to have about the same probability to be eventually rolled back if selected for the execution. In other words, given a simulated time interval $\mathrm{SI} = [min\_ts, min\_ts + \mathrm{SW}]$, if an adequate value for SW is selected then all the events belonging to S and having timestamp within SI are likely to have about the same probability to be eventually rolled back as the event with the minimum timestamp min_ts. This intuition is supported by some empirical results reported in (Quaglia and Cortellessa 2000).

In our context, the interval SI can be used to identify a set of non-executed events such that the aggressive execution of whichever event belonging to this set is likely to produce the same aggressiveness effects on rollback as the non-executed event with the minimum timestamp. We call this set as Low Aggressiveness Effects Set ($LAES$). This set can be formally defined as:

$$LAES = \{x \mid (x \in S) \wedge (timestamp(x) \in \mathrm{SI})\} \qquad (2)$$

$LAES$ allows us to manage the previous tradeoff in practice since it provides multiple events, originating low aggressiveness effects, among which the event to be selected for the execution should be determined in order to minimize the effects of risk as modeled by equation (1). This is the final objective of the ARES scheduling algorithm to be presented in the next section.

Finally we underline that the notion of SI allows also the simplification of the model in (1) when restricting the model itself to the events belonging to $LAES$. Specifically, the probability values $P(x)$ associated with all the events x belonging to $LAES$ can be approximated with a single probability value that we denote as P. Therefore, for any event x belonging to $LAES$, the model can be rewritten as:

$$C(x) = P \cdot fo(x) \qquad (3)$$

We refer to the model in (3) as the *approximated model*. The validity of this model obviously derives from the adequacy of the value of SW determining the interval SI which, in its turn, defines the set $LAES$.

3.3 The Algorithm Structure

From considerations in previous section it comes out that the structure of ARES should be: (i) to compute for any event belonging to $LAES$ the value of the cost function in equation (3) and then (ii) to select for the execution the event associated with the minimum cost. Note that the approximated model in (3) is such that the identification of the event in $LAES$ associated with the minimum cost means in practice identifying the event associated with the minimum fan-out, therefore we can derive the final structure of ARES as shown in Figure 2.

There exists the possibility that multiple events in $LAES$ have minimum fan-out. It this case, the selection in line 4 of the algorithm should resolve ties according to timestamp values. Specifically, among multiple events, the one with the minimum timestamp should be selected following the classical approach underlying the LTF algorithm.

Two points still remain to be touched. The first is how to select the length of the interval SI (that is, the length of the scheduling window SW); this point will be shortly discussed below. The second is how to compute (predict) the fan-out of the events in $LAES$, which is needed in line 4 of the algorithm; this point will be discussed in the next subsection.

For what concerns SW, and therefore the scheduling interval SI, it is intuitively true that the value to be selected in order to ensure the validity of the approximated cost model in (3) could change while the simulation progresses. This is because the rollback behavior of the simulation depends on several parameters that do not necessarily reach a steady state. While for a given real time interval it would be better keeping a narrow SW, during a different real time interval it could be kept larger. Tuning dynamically SW to the best suited value contributes to high flexibility of the scheduling decision without invalidating the approximated cost model, and thus without leading to an increase of the effects of aggressiveness on rollback. Large values for SW could lead ARES to select for the execution an event x whose timestamp is sensitively larger than min_ts. Depending on the instant this happens, it could actually impact (or not) the validity of the approximated model since $P(x)$ could be sensitively larger (or not) than the probability for the event in $LAES$ having the minimum timestamp to be eventually rolled back. Adequate dynamical tuning of SW should overcome this problem.

Similarly to the GS algorithm in (Quaglia and Cortellessa 2000), the length of SW can be recalculated as a function of the variations of a reference performance parameter, whose value can be monitored on-line. More precisely, an initial value of zero is selected for SW; then SW is increased/decreased depending on variations of the performance parameter. The authors of the GS algorithm discussed how SW must be a global parameter in that it must have the same value on all the processors. If this does not happen, then no effective monitoring of the real impact of its value on performance can be implemented.

This type of tuning allows ARES to adapt the scheduling decision to the (dynamic) behavior of the overlaying application. In addition, if the tuning leads the value zero to be selected for SW, then ARES behaves as the classical LTF algorithm. This points out how ARES has the potential to recover towards a standard scheduling behavior in case performance loss is noted.

In our implementation of ARES we have selected the *event rate*, namely committed events per time unit, as the

```
if $S \neq \emptyset$
then
    <compute the set $LAES = \{x \mid (x \in S) \wedge (timestamp(x) \in \mathrm{SI})\}$ >;
    <select for execution an event $e \in LAES$ such that $\forall e' \in LAES\ fo(e) \leq fo(e')$ >
else <no action>
```

Figure 2: ARES Structure

reference performance parameter for the adaptation of the length of SW, which is realized as follows. We suppose the simulation execution as divided in periods, each period consisting of a fixed number of executed events. SW is initially set to zero (therefore ARES initially behaves like LTF). At the end of every period statistics on the event rate are collected from each processor. If the event rate does not decrease, then SW is increased by a fixed quantity α_{inc}. Otherwise, there is evidence that the value of SW adopted in the last period may be too large thus invalidating the approximated model in (3) and originating an increase of aggressiveness effects. In this case SW is decreased by $\alpha_{dec} = h \times \alpha_{inc}$ with $h > 0$ (in case SW is less than α_{dec}, it is set to zero). The step for the decrease with $h > 1$ allows quick recovery towards a classical simulation asset, namely LTF scheduling, if successive decreases of the event rate are noted. In our implementation we select for h the value 2. The idea behind the dynamical recalculation is to try to provide large values for SW (that should allow more flexible decisions as a function of the effects of risk), provided that the assumption underlying the approximated model in (3) is verified (i.e. no increase of aggressiveness effects originating performance loss is noted).

For what concerns the value of α_{inc}, it could be selected as a function of several application specific features such as event density, timestamp increment distribution functions and so on. In (Quaglia and Cortellessa 2000) the following general rule has been introduced: $\alpha_{inc} = T/10$ with T being the average value among the means of the distribution functions for the timestamp increment. More sophisticated, application-tailored solutions could be however envisaged.

3.3.1 Computing the Fan-Out

Line 4 of the algorithm requires the knowledge of the fan-out of the events belonging to $LAES$. In most simulations it is usual to have multiple event types, each one associated with a given code. All the events belonging to a specific event type have the same expected fan-out value. If the code of an event type is deterministic, then the fan-out value for the events of that type is deterministic as well, and it can be determined off-line by the code designer, in a totally transparent way to the user. This value can be used by ARES. The same thing happens for the case of non deterministic code containing branches and/or loops that have no effects on the fan-out. Instead for the case of code structure with branches or loops that can have a real effect on the fan-out of the associated event type, the expected fan-out must be estimated on-line. This could be done using samples related to the recently executed portion of the simulation. The estimate can be computed infrequently, in order to not produce probing effects on the simulation.

3.4 Considerations on Long-Term Effects of ARES

Any event undone by rollback could originate a *rollback tree*. In Figure 3 two possible rollback trees due to the undoing of the events c and f, depicted in the example in Figure 1, are shown. Each tree contains all the events that have been undone consequently to the undoing of the event at the root of the tree. For example, *tree(c)* shows that, after c has been undone, two LPs roll back due to antimessage receipt. These LPs undo, respectively, seven and four events. Some of these events require antimessage sending that, in its turn, produces rollback on other LPs and so on. Undoing the event f generates a similar rollback tree (also this tree is shown in Figure 3).

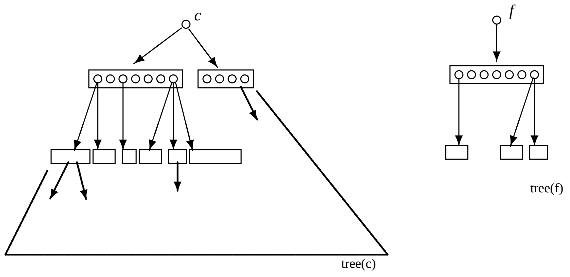

Figure 3: Rollback Trees

It is straightforward that the real rollback tree can be drawn only after the rollback occurrence. This is because rollback spreading depends on relative positions of the LVTs of the LPs. In other words, an antimessage produces a rollback (or not) depending on its timestamp and on the value of the LVT of the recipient LP at the time the antimessage is received.

However, independently of LVT positions, the fan-out associated with an event determines the n-arity of the root of the rollback tree since as many antimessages have to be sent as the fan-out value. The depth of a rollback tree with a large root n-arity is likely to be larger than the one of a tree with a small root n-arity. This is because many antimessages are more likely to let the rollback survive than few antimessages. The larger the difference between the n-arities, the higher the probability that this presumption reveals true. This is a support to the idea that ARES should be likely to reduce rollback for applications with high fan-out variance among distinct event types.

Finally we outline that ARES tends to erase *entire* rollback trees. Specifically, delaying any risky event x has the effect to make x more likely to be committed because its timestamp distance from the GVT does not increase (this is due to the monotonic increase of the GVT). In other words, a throttling effect on the event x is originated. This means a reduction of the probability $P(x)$ for x to be eventually undone that, according to the cost model in (1), will further reduce the effects of risk on rollback and on communication cost.

4 PERFORMANCE DATA

In this section we report a performance study of ARES conducted using a parameterized synthetic benchmark. Prior to presenting the results, we describe the testing environment, the benchmark itself and the performance parameters we have observed.

4.1 Testing Environment

The experiments reported in this paper were all performed on a cluster of 4 PCs Pentium II 300 MHz (128 MB RAM) running LINUX as operating system, interconnected by a high speed Myrinet switch based on wormhole technology. This type of architecture is actually an emerging one for parallel applications due to cost vs performance reasons and also to expansibility/modifiability.

Any PC is connected to the Myrinet switch through an interface implemented on a card consisting of a LANai processor equipped with local memory and supports for DMA. The LANai's memory is mapped into the address space of the host PC, therefore it can be accessed either directly or using DMA. The LANai processor runs a *control program* that performs send and receive operations. This program can be designed according to requirements of the specific application. Depending on the structure of the control program and of the associated message passing layer run at the host PC, messages at the receiver side can be buffered into the host PC memory or into the memory on board of the interface card and then transferred on demand into the host memory. We have developed a high speed layer, namely Minimal Fast Messages (MFM), tailored for optimizing the delivery delay of small size messages. This layer results therefore well suited for parallel simulation applications where the amount of data associated with messages/antimessages transmission is typically small. In MFM the buffering at the receiver side is done into the on board memory of the interface card.

In our Time Warp system, message exchange among LPs hosted by the same machine does not involve operations of the MFM layer. There is an instance of the Time Warp kernel on each processor. The kernel manages the local event list (resulting as the logical collection of the input queues of the local LPs) and schedules LPs for running according to the selected scheduling algorithm. Memory space for new entries into the input and output queues of the LPs is allocated dynamically using classical `malloc()` calls. Therefore there is no pool of pre-allocated buffers. The same dynamical approach has been used for entries of the stack storing saved state vectors. The cancellation phase is implemented following the aggressive policy (Gafni 1985). Fossil collection is executed periodically.

4.2 Benchmark and Performance Parameters

In order to test the effectiveness of ARES, we have used the synthetic benchmark known as PHOLD model, originally presented in (Fujimoto 1990). PHOLD consists of a fixed number of LPs and of a set of jobs (messages) circulating among the LPs. Both the routing of jobs among the LPs and the timestamp increments are taken from some stochastic distributions. Although a set of standard benchmarks for parallel discrete event simulation does not exist, PHOLD is in practice one of the most used ones for two main main reasons: (i) its parameters (e.g. event execution time, size of the state vectors, etc.) can be easily modified, (ii) it usually shows a rollback behavior similar to many other synthetic benchmarks and to several real world models.

The PHOLD model we have considered is composed of 32 homogeneous LPs evenly distributed among the 4 PCs of the cluster. There are two distinct job (message) types, namely A and B. When an event associated with a job of type A is executed, the effect is the production of n new jobs such that: (i) one new job is of type A, (ii) the remaining $n-1$ jobs are of type B. Instead, the execution of an event associated with a job of type B does not produce any new event. Therefore, events associated with jobs of type B have fan-out equal to zero, while n indicates the fan-out of the events associated with jobs of type A. This kind of event generation rules are depicted in Figure 4. Timestamp increments associated with new jobs are selected from an exponential distribution with mean 10 simulated time units.

Any value of n ranging from one to several dozens is likely to produce stable system behavior in that buffers are likely to not overflow and, at the same time, the message population (i.e. the amount of messages circulating in the system) does never get lower than the amount of messages of type A initially inserted in the system. Obviously larger

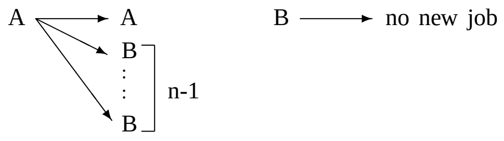

Figure 4: Job Generation Rules

values for n determine larger variance for the fan-out of simulation events. In our experiments we have selected an initial job population of 1 job of type A per LP and we have varied n from 1 to 9. Given previous job generation rules, when $n = 1$ only jobs of type A circulate in the system.

As respect to job (message) routing, we have selected an uniform approach, that is any new job is equally likely to be sent to any LP independently of the job type. The processing time for any event has been fixed at about 140 microseconds. State saving is performed before the execution of any new event and the cost to copy the state vector has been fixed at about 70 microseconds.

We report measures related to the following performance parameters:

- The *rollback frequency* that is, the ratio between the number of rollbacks and the total number of executed events, and the *average rollback length* that is, the average number of undone events by each rollback occurrence. These two parameters allow us to point out whether (and how) different scheduling algorithms determine changes in the final rollback pattern and thus in the amount of rollback.
- The *antimessages frequency* that is, the number of antimessages per time unit. This parameter points out the possible saving of communication cost of one scheduling algorithm vs another. We decided to consider this parameter in the analysis because ARES could actually produce a reduction of the amount of antimessages that have to be sent. This parameter allows us to quantify this possible reduction.
- The *event rate* that is, the number of committed events per time unit. This parameter indicates how fast is the simulation execution with a given scheduling algorithm, it is therefore representative of the final performance perceived.

As stated before, the event rate acts also as a reference performance parameter for the tuning of the scheduling window SW in ARES. For the present experiments, the period for the recalculation of SW is fixed at 10000 executed events per processor. When the period expires at a given processor that we identify as the master for the recalculation, this processor collects statistics on the performance parameter, recalculates the value of SW and notifies it to the other processors.

We report the average observed values of previous parameters, computed over 20 runs all done with different seeds for the random number generation. At least 2×10^6 committed events have been simulated in each run. As reference scheduling algorithm to point out the effectiveness of ARES we have selected LTF.

4.3 Results

The obtained results, reported in Figures 5 through 8, point out that when the value of n is set to 1, LTF and ARES produce exactly the same performance. This is an expected behavior since $n = 1$ means that only jobs of type A (with fan-out equal to 1) circulate in the system. Therefore ARES always selects for the execution the non-executed event with the lowest timestamp.

From the results in Figure 5 and in Figure 6 we have that for any value of n larger than 1, LTF shows lower rollback frequency, up to 26%, but exhibits longer average rollback length, up to 30%. Multiplying the rollback frequency by the average rollback length we get the so called *efficiency* of the simulation. This parameter represents the probability for whichever event to be not eventually rolled back; it is therefore representative of the amount of rollback. By the obtained results we get that ARES allows slightly more efficient execution.

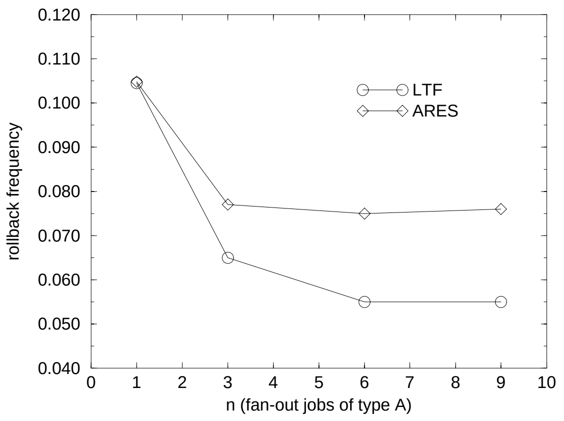

Figure 5: Rollback Frequency vs n

Beyond rollback behavior in terms of efficiency, a relevant result comes out from the strong reduction of the number of antimessages under ARES when n is larger than 1. In particular, plots in Figure 7 show that the number of antimessages per time unit is reduced up to 20% when ARES is used. As already discussed this has the potential for a strong reduction of the real communication cost due to a reduction of the antimessage overhead and also to a reduction of the amount of revoked notification messages.

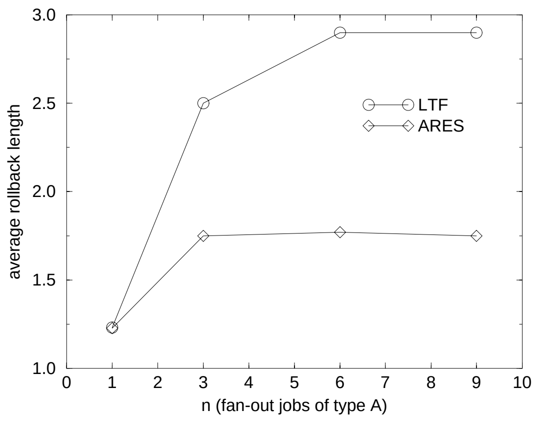

Figure 6: Average Rollback Length vs n

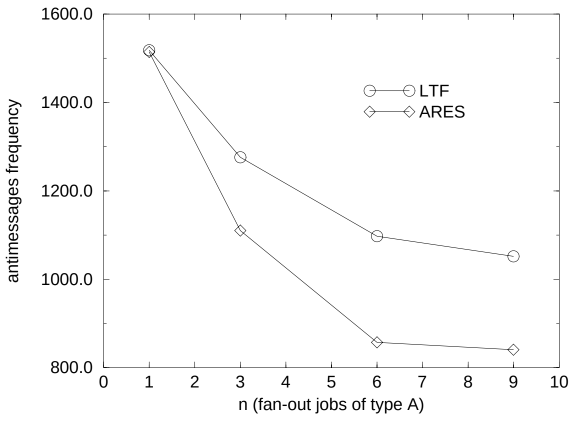

Figure 7: Antimessages Frequency vs n

The slightly higher efficiency, together with the strong reduction of the communication cost when n is larger than 1 point out how ARES actually produces better scheduling decisions (in terms of real effects of aggressiveness and risk on performance) when there exists at least minimum variance for the fan-out of the events. Plots in Figure 8 show the maximum performance gain of ARES is about 10% and is noted for $n = 6$ and $n = 9$.

As last consideration we note that while n increases up to 6, the event rate under both algorithms grows due to an increase in the efficiency. Instead, for larger values of n the efficiency remains stable therefore we get a sensible decrease of the event rate due to higher cost of the event list management originated by larger job population.

5 SUMMARY

In this paper we have presented a scheduling algorithm for the selection of the next LP to be run on a processor in Time Warp simulations. The algorithm aims at reducing the effects of both aggressiveness and risk on performance. It

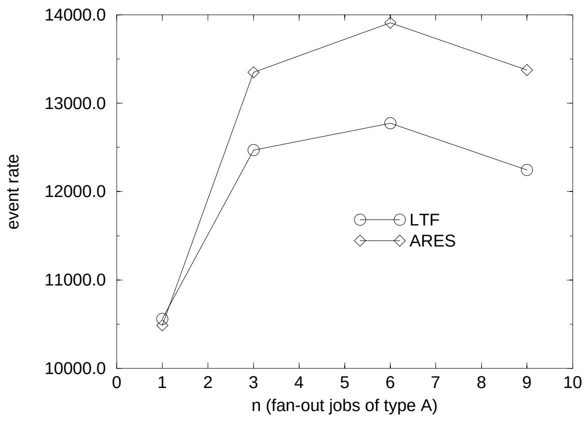

Figure 8: Event Rate vs n

differs from most previous solutions in that they consider exclusively the effects of aggressiveness and discard those associated with risk. We have tested the effectiveness of the algorithm using a classical synthetic benchmark. The obtained results point out the viability of our solution in the reduction of the amount of rollback of the simulation (due to both aggressiveness and risk) and the real communication cost associated with risk. The impact of this reduction is an increase of the speed of the simulation execution. The performance gain is relevant for the cases of non-minimal variance of the amount of new events to be notified by the execution of different event types. This algorithm results therefore suited for all those simulations exhibiting this feature.

REFERENCES

Ball, D. and S. Hoyt. 1990. The adaptive Time-Warp concurrency control algorithm. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 174-177.

Dickens, P.M. and P.F. Reynolds Jr. 1990. SRADS with local rollback. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 161-164.

Fan, L., P. Cao, J. Almeida and A.Z. Broden. 1998. Summary cache: a scalable wide-area web cache sharing protocol. In *Proceedings of the ACM Sigcomm'98*, 254-265.

Ferscha, A. 1995. Probabilistic adaptive direct optimism control in Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS'95)*, 120-129.

Ferscha, A. and J. Luthi. 1995. Estimating rollback overhead for optimism control in Time Warp. In *Proceedings of the 28th Annual Simulation Symposium*, 2-12.

Fujimoto, R.M. 1990. Performance of Time Warp under synthetic workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1).

Gafni, A. 1985. Space management and cancellation mechanisms for Time Warp. *Tech. Rep. TR-85-341*, University of Southern California, Los Angeles (Ca,USA).

Jefferson, D. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404-425.

Lin, Y.B. and E.D. Lazowska. 1991. Processor scheduling for Time Warp parallel simulation. In *Advances in Parallel and Distributed Simulation*, pp.11-14.

Lubachevsky, B., A. Weiss and A. Shwartz. 1989. Rollback sometimes works ... if filtered. In *Proceedings of the 1989 Winter Simulation Conference*, 630-639.

Madisetti, V.K., D.A. Hardaker and R.M. Fujimoto. 1993. The MIMDIX environment for parallel simulation. *Journal of Parallel and Distributed Computing*, 18:473-483.

Palaniswamy. A.C. and P.A. Wilsey. 1994. Scheduling Time Warp processes using adaptive control techniques. In *Proceedings of the 1994 Winter Simulation Conference*, 731-738.

Preiss, B.R., W.M. Loucks and D. MacIntyre. 1994. Effects of the checkpoint interval on time and space in Time Warp. *ACM Transactions on Modeling and Computer Simulation*, 4(3):223-253.

Quaglia, F. 2000. A state-based scheduling algorithm for Time Warp synchronization. In *Proceedings of the 33rd Annual Simulation Symposium*, 14-21.

Quaglia, F. and V. Cortellessa. 2000. Grain sensitive event scheduling in Time Warp parallel discrete event simulation. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'00)*, 173-180.

Reynolds, P.F. Jr. 1988. A spectrum of options for parallel simulation. In *Proceedings of the 1988 Winter Simulation Conference*, 325-332.

Ronngren, R. and R. Ayani. 1994. Service oriented scheduling in Time Warp. In *Proceedings of the 1994 Winter Simulation Conference*, 1340-1346.

Som, T.K. and R.G. Sargent. 1998. A probabilistic event scheduling policy for optimistic parallel discrete event simulation. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, 56-63.

Sokol, L.M., D.P. Briscoe and P.A. Wieland. 1988. MTW: a strategy for scheduling discrete events for concurrent execution. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 34-42.

Srinivasan, S. and P.F. Reynolds Jr. 1998. Elastic time. *ACM Transactions on Modeling and Computer Simulation* 8(2):103-139.

Steinman, J. 1993. Breathing Time Warp. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'93)*, 109-118.

Turner, S.J. and M.Q. Xu. 1992. Performance evaluation of the bounded Time Warp algorithm. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92)*, 117-126.

Wessel, D. and K. Claffy. 1998. Internet cache protocol (ICP). Version 2, 1998. `<http://ds.internic.net/rfc/rfc2186.txt>`.

AUTHOR BIOGRAPHIES

VITTORIO CORTELLESSA received his Laurea degree in Computer Science from University of Salerno in 1991 and his Ph.D. degree in Computer Engineering from University of Rome "Tor Vergata" in 1995. He has been visiting research associate at CERC (West Virginia University) in 1994, and post-doc fellow at ESA (Esrin, Rome) in 1996. He is currently research assistant professor with the Computer Science Department of West Virginia University and research associate with the Department of Computer Science Systems and Production of University of Rome "Tor Vergata". His research interests include performance modeling and evaluation of software/hardware systems, parallel simulation, software requirement specifications engineering. He serves as a referee for several international conferences and journals.

FRANCESCO QUAGLIA received the Laurea degree in electronic engineering in 1995 and the Ph.D. degree in computer engineering in 1999 from the University of Rome "La Sapienza". From summer 1999 to summer 2000 he held an appointment at CNR ("Consiglio Nazionale delle Ricerche" - Italy). Currently he is an assistant professor at the University of Rome "La Sapienza". His research interests include parallel discrete event simulation, parallel computing, fault-tolerant programming and performance evaluation of software/hardware systems. He regularly serves as a referee for several international conferences and journals. He was invited to serve on the program committee of the 14th and 15th editions of the Workshop on Parallel and Distributed Simulation (PADS).