Grain Sensitive Event Scheduling in Time Warp Parallel Discrete Event Simulation

Francesco Quaglia
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, 00198 Roma, Italy
quaglia@dis.uniroma1.it

Vittorio Cortellessa
Dipartimento di Informatica Sistemi e Produzione
Università di Roma “Tor Vergata”
Via di Tor Vergata, 00133 Roma, Italy
cortelle@info.uniroma2.it

Abstract
Several scheduling algorithms have been proposed to determine the next event to be executed on a processor in a Time Warp parallel discrete event simulation. However, none of them is specifically designed for simulations where the execution time (or granularity) for different types of events has large variance. In this paper we present a grain sensitive scheduling algorithm which addresses this problem. In our solution, the scheduling decision depends on both timestamp and granularity values with the aim at giving higher priority to small grain events even if their timestamp is not the lowest one (i.e. the closest one to the commitment horizon of the simulation). This implicitly limits the optimism of the execution of large grain events that, if rolled back, would produce large waste of CPU time. The algorithm is adaptive in that it relies on the dynamic recalculation of the length of a simulated time window within which the timestamp of any good candidate event for the scheduling falls in. If the window length is set to zero, then the algorithm behaves like the standard Lowest-Timestamp-First (LTF) scheduling algorithm. Simulation results of a classical benchmark in several different configurations are reported for a performance comparison with LTF. These results demonstrate the effectiveness of our algorithm.

1 Introduction
In a parallel discrete event simulator, each part of the simulated system is modeled by a distinct logical process (LP) which is basically a sequential discrete event simulator having its own simulation clock, namely Local Virtual Time (LVT), its own event list and its own state variables [4]. The execution of any simulation event at an LP usually modifies its state and possibly produces new events to be executed at later simulated time. The notification of new events among distinct LPs takes place through the exchange of messages carrying the content and the occurrence time, namely timestamp, of the event. Synchronization mechanisms are used to ensure a timestamp ordered execution of events at each LP, which is a sufficient condition for the correctness of simulation results.

In Time Warp synchronization [7], the simulation progress of any LP is “optimistic” in that the LP executes its events as soon as they are available, without the guaranty of no timestamp order violation. If a timestamp order violation is detected, then a rollback procedure is executed for recovering the LP state to its value prior the violation. While rolling back, the LP “cancels” the events produced during the rolled back part of the simulation; the event cancellation possibly leads other LPs to rollback. It is widely recognized that one of the essential factors having an impact on the performance of Time Warp synchronization is the scheduling algorithm for the selection of the next event to be executed on a processor. We recall that the need for a scheduling algorithm arises as for simulations of large and/or complex systems it is extremely likely that any processor is responsible for the execution of more than one LP and several LPs hosted by the same processor may simultaneously have at least one non-executed event in their event lists.

Currently the term “good” for a scheduling algorithm is interpreted as the capability of the algorithm to allow fast completion of the simulation by producing a low amount of rollback (i.e. a low amount of rolled back events), which typically means low rollback frequency and short rollback length. However, this perspective becomes inadequate when dealing with Time Warp simulations having high variance of the event granularity, that is, high variance of the event execution time; this is typical of several real world simulations, such as battlefield simulations or simulations of mobile communication systems. In this context, the term “good” should be interpreted as the capability of the scheduling algorithm to allow fast completion by producing a low amount of rollback and also by guaranteeing that the majority of the rolled back events are fine grain ones. The combination of these two features will allow a reduction of the waste of CPU time due to rolled back events, compared to scheduling algorithms which attempt only to reduce the amount of rollback without taking granularity features into account.

In this paper we present a scheduling algorithm, namely Grain Sensitive (GS), specifically designed for simulations with high variance of the event granularity. In this as-
pect GS differs from all previous scheduling algorithms. Actually our solution is a modification of the Lowest-Timestamp-First algorithm (LTF) [8] which is recognized as a standard solution for the scheduling problem. LTF always schedules for the execution the non-executed event with the minimum timestamp, say \(e\), under the implicit assumption that it has the lowest probability to be rolled back in the future of the simulation execution (this is because it is the closest one to the commitment horizon). Instead, GS delays the scheduling of \(e\) each time there is a non-executed event \(e'\) (of an LP distinct from the owner of \(e\)) such that: (i) the granularity of \(e\) is larger than that of \(e'\) and (ii) the distance between the timestamps of \(e\) and \(e'\) is within a Scheduling Window (SW). The rationale behind this solution is as follows. If SW is selected appropriately, then \(e\) and \(e'\) will have about the same probability to be eventually rolled back if scheduled for the execution, therefore:

(a) scheduling \(e'\) prior to \(e\) is likely to not increase the amount of rollback of the simulation;

(b) delaying the execution of \(e\) might decrease the probability that \(e\) will be eventually rolled back (this is the classical positive throttling effect [1, 2, 19]); as \(e\) has large granularity, we get a reduction of the probability that large grain events are rolled back, thus keeping low the waste of CPU time.

The combination of features in points (a) and (b) leads GS to be a “good” algorithm when dealing with simulations having high variance of the event granularity. The key problem to address in order to ensure features in points (a) and (b) is the selection of an adequate size for the scheduling window SW. Too large values for SW may lead feature in point (a) to be not ensured as \(e'\) will actually have higher probability than \(e\) to be eventually rolled back (this will possibly degrade performance due to an increase in the amount of rollback of the simulation). On the other hand, too small values for SW may lead feature in point (b) to be not ensured as there is low probability to produce the throttling effect on \(e\), even if \(e\) has very large granularity (this will not reduce the waste of CPU time compared to the classical LTF algorithm). To tackle this problem GS is adaptive in that it relies on the dynamic recalculation of SW based on the monitoring of the real waste of CPU time which is a measure of both the amount of rollback of the simulation and the average granularity of the rolled back events. As an extreme, adaptiveness may lead the value zero to be selected for SW; if this occurs then GS behaves like LTF. This outlines the generality of GS, which has the capability to produce at least the same performance of a standard algorithm.

We report simulation results of a classical benchmark in several different configurations demonstrating the effectiveness of our solution. These results show that GS provides performance improvements in every case considered.

The remainder of the paper is organized as follows. In Section 2 we report a brief overview of existing scheduling algorithms. GS is described in Section 3 together with the technique for the adaptive selection of SW. The performance data are reported in Section 4.

2 Background

The standard solution for the scheduling problem is the Lowest-Timestamp-First algorithm (LTF) [8] which always schedules as next event to be executed on a processor the one with the minimum timestamp. 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 (i.e. the commit horizon) of the simulation \(i^1\). If this presumption reveals true, as in most simulations, then LTF keeps low the amount of rollback, thus producing good performance. Another scheduling algorithm, namely Lowest-Local-Virtual-Time-First (LLVTF) [10], gives higher priority to LPs having lower simulation clocks (i.e. lower LVTs). 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, thus reducing the probability for any LP to induce a timestamp order violation in any other LP involved in the simulation. In [9] 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 events of the LPs having higher values of their useful work. The model for the useful work developed by the authors assumes that all the events have the same granularity. A rather different solution, namely Service Oriented scheduling (SO), is presented in [15]. The idea behind this solution is to try to produce and deliver as soon as possible events not yet produced that will have the lower timestamps. This is done in order to deliver promptly those 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 event whose execution will produce the event with the minimum predicted timestamp. A Probabilistic scheduling algorithm (P) has been presented in [17]. The consideration at the basis of this algorithm is that low amount of rollback can be obtained if the event scheduled for the 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 scheduled for the execution. Finally, in [14] a State Based scheduling algorithm (SB) has been presented. In this algorithm, the schedul-
ing 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.

None of the previous algorithms is designed to maximize the performance for the case of simulations with high variance of the event granularity. This is because the event granularity either is not taken into account at all, or is assumed to have no variance like for the case of AC. However there exist several important classes of simulations in which the execution of different types of events actually takes very different amounts of CPU time. Examples are simulations of battlefields and of mobile communication systems. The GS scheduling algorithm we present in this paper copes with high event granularity variance, thus being well suited for previous classes of simulations.

3 Grain Sensitive Scheduling

This section describes the GS scheduling algorithm. We proceed along the following line. We first provide the basic ideas underlying the algorithm. Then we describe the algorithm structure into details and present the solution for the adaptive tuning of the scheduling window SW. Finally we discuss the relation between GS and classical throttling algorithms.

3.1 Basic Ideas

Consider the example shown in Figure 1 involving three LPs, namely LP₁, LP₂ and LP₃. Assume that these three LPs are hosted by the same processor P and no other LP is hosted by P. Suppose the event list of LP₁ contains two non-executed events a and b, the event list of LP₂ contains one non-executed event c and the event list of LP₃ contains two non-executed events d and e. The timestamps associated with these events are those shown in Figure 1. Furthermore, suppose the width of the rectangular box associated with a simulation event is representative of the event granularity. We denote the granularity of an event x as G(x). In our example, the event a has granularity G(a) which is four times the granularity G(c) of the event c.

The correctness criterion the simulation relies on requires that each LP must execute its events in the order of their timestamps (2). We call the set of non-executed events that could be scheduled without violating the correctness criterion as the Scheduling Candidates set, denoted as SC. The set SC contains for any LP the lowest timestamp with the event that is not yet executed. Furthermore, we denote as min(SC) the event e ∈ SC having the minimum timestamp value. If several events have the minimum timestamp value then min(SC) denotes whichever of those events. For the example in Figure 1 we have SC = {a, c, d} and min(SC) = a.

IDEA-1. If the timestamp of an event e ∈ SC distinct from min(SC) is not so different from the timestamp of min(SC), then min( SC ) and e are likely to have the same probability to be eventually rolled back as they have about the same distance from the Global-Virtual-Time (GVT) of the simulation. Thus, we get that there may exist a simulated time interval I(SW) defined as follows:

\[
I(SW) = [ts(min(SC)), ts(min(SC)) + SW]
\]  

such that all the events in SC with timestamp within I(SW)
have about the same probability as min(SC) to be eventually rolled back if currently scheduled for the execution. All these events are good candidates for the scheduling as the execution of any of them is likely to not decrease the probability to perform productive simulation work compared to

As pointed out before, we refer to SW as scheduling window. The value of SW adequate for a given simulation (i.e. guaranteeing that all the events with timestamp within I(SW) actually have about the same probability to be eventually rolled back if currently scheduled for the execution) depends on features proper of the simulation, such as number of LPs, how they interact, etc. Each simulation has its adequate SW value which could also change in the lifetime of the simulation. For the example in Figure 1, we get that if at the time the scheduling decision must be taken the adequate value for SW is 3 simulated time units, then both a and e are good candidates for the scheduling.

The relevance of IDEA-1 for the scheduling problem is as follows. Having a set of events considered as good candidates for the execution increases the flexibility of the scheduling decision so that the decision can be aimed at improving the performance by exploiting granularity features of the good candidate events.

IDEA-2. The second idea is related to the classical notion of throttling in Time Warp synchronization. Specifically, given a non-executed event e, we have that the probability for e to be eventually rolled back if executed at real time t is higher than, or at least equal to, the probability to be eventually rolled back if executed at real time t + Δt. This is because:

\[ ts(e) - GVT[t] \geq ts(e) - GVT[t + \Delta t] \]  

In other words, as the GVT value does not decrease in real time, the difference between the timestamp of the event e and the GVT does not increase in real time (i.e. the distance of the event from the commitment horizon of the simulation does not increase); therefore the probability for the event e to be eventually rolled back is likely to decrease if its execution is delayed.

The relevance of IDEA-2 for the scheduling problem is as follows. If we can delay the execution of an event e which has very large granularity without increasing the whole amount of rollback, compared to the classical LTF scheduling algorithm, then we can get an increase in the performance.

Both IDEA-1 and IDEA-2 form the basis of the GS scheduling algorithm presented in the next subsection.

3.2 The Scheduling Algorithm

Let us assume at a given real time instant t a scheduling decision must be taken on processor P. Also, assume that SW is the adequate value of the scheduling window at real time t. According to IDEA-1, SW determines a set of Good Candidate events for the scheduling, namely \( GC \), containing at least \( min(SC) \). GC is formally defined as:

\[ GC = \{e \in SC | ts(e) \in I(SW^*)\} \]  

Let us denote as \( P_r(e,t) \) the probability for the event \( e \in GC \) to be eventually rolled back if scheduled for the execution at real time t. Adequacy of the scheduling window \( SW^* \) implies that, for any pair of events \( e \) and \( e' \) belonging to GC, \( P_r(e,t) \approx P_r(e',t) \). We approximate these probability values with \( P_r(t) \) representing the probability for any event belonging to GC to be eventually rolled back if scheduled for the execution at real time t.

We can now build the following simple model \( M(e,t) \) for the waste of CPU time associated with the scheduling of the event e belonging to GC for the execution at real time t:

\[ M(e,t) = P_r(t)G(e) \]  

where \( G(e) \) is the granularity of e. Note that this model neglects the effects any scheduling action will produce on the progress of the simulation. It takes into account only the waste of CPU time predictable at time t as a function of the granularity of the events considered as good candidates for the scheduling. In this aspect, it shares the same limitation of the scheduling algorithms discussed in Section 2, except the one in [15], as these algorithms do not take any future effect of the outcome of the current scheduling decision into account (3).

The rationale behind the GS algorithm is to schedule for the execution at time t the event e belonging to GC which minimizes the value of the function in (4). This simply implies the scheduling of the event \( e \in GC \) with the lowest granularity value. Compared to LTF, this scheduling decision will produce the following two positive effects each time the scheduled event e has not the minimum timestamp in the set GC:

- the waste of CPU time due to the outcome of the scheduling decision at time t is reduced; this is because LTF would schedule for the execution \( min(SC) \) with granularity \( G(min(SC)) \) > \( G(e) \), therefore \( M(min(SC),t) > M(e,t) \). In other words, the waste of CPU time produced by the scheduling decision according to the model in expression (4) is larger for LTF than for GS;

- a throttling effect is produced on \( min(SC) \) whose execution is delayed from real time t to real time \( t + \Delta t \); if \( min(SC) \) has large granularity then we get a reduction of the probability that large grain events are rolled back, thus producing a further reduction of the waste of CPU time compared to LTF.

The first effect is a direct effect due to the minimization of the model for the waste of CPU time in (4). The second

\[ 3 \]  

In the algorithm in [15] the effects of the scheduling decision on the progress of the simulation are taken into account at some extent. This is because the scheduling decision relies on (predicted) timestamp values of future events not yet produced.
effect is an indirect effect due to the delay induced for the execution of large grain events.

Considering the example in Figure 1, if we assume the adequate scheduling window is \( SW^* = 4 \) then GS favors the fine grain event \( e \) compared to the larger grain one \( a \).

In Figure 2, the GS scheduling algorithm is shown. Although not discussed before, there may exist several events in \( GC \), with the same granularity value, minimizing the cost model in (4) (see Line 5 of the algorithm). If this occurs, GS schedules from among these events the one with the minimum timestamp (see Line 6), following the classical policy underlying LTF.

The key problem in order to guarantee the feasibility of the model in (4) underlying the algorithm is how to select the adequate value \( SW^* \) for the scheduling window. This issue will be tackled in the following subsection. It is important to note that if the selected value for \( SW^* \) is zero, then the GS scheduling algorithm behaves similarly to the classical LTF algorithm. In this case, the major difference is that if \( GC \) contains several events, then GS gives the highest priority to the event belonging to \( GC \) with the lowest granularity value, while LTF randomly selects in \( GC \) the event which must be executed. This slightly different behavior of GS will possibly favor the performance, as compared to LTF, by producing throttling on large grain events. The similarity in the behaviors of GS and LTF when the value of \( SW^* \) is set to zero will be exploited in the technique for the adaptive recalculation of the value of \( SW^* \).

Finally, we recall that the event granularity could be either deterministic or of a random nature. In the second case, an estimation system for predicting the expected granularity of the events in \( GC \) should be coupled with the GS algorithm.

### 3.3 Adaptive Tuning of the Scheduling Window

We base the adaptive selection of the value of the adequate scheduling window \( SW^* \) on the monitoring of the average waste of CPU time per event, namely \( WT \). From a mathematical point of view \( WT \) is defined as the following product of two quantities:

\[
WT = P_r \cdot \tilde{G} \tag{5}
\]

The former quantity, namely \( P_r \), is the ratio between the number of rolled back events and the total number of executed simulation events (committed plus rolled back). \( P_r \) expresses the probability for whichever executed event to be eventually rolled back, it is therefore representative of the amount of rollback of the simulation (\(^4\)). The latter quantity, namely \( \tilde{G} \), is the average granularity of the executed events (committed plus rolled back). If we denote with \( EX \) the set of the executed events, then \( \tilde{G} \) is expressed as:

\[
\tilde{G} = \frac{\sum_{e \in EX} G(e)}{EX} \tag{6}
\]

The LTF scheduling algorithm produces some values for \( P_r \) and \( G \). The objective of the GS algorithm is to produce a probability value \( P_r \) similar to that produced by LTF (this should be the outcome of IDEA-1) and an average granularity value \( \tilde{G} \) lower than that produced by LTF (this should be the outcome of IDEA-2) as under GS the set of executed events \( EX \) should contain less rolled back large grain events due to the throttling effect on them. This points out the adequacy of \( WT \) as expressive parameter of the performance of GS, as compared to LTF, since it accounts for the two quantities directly affected by the scheduling decision performed by GS. Specifically, the difference of the \( WT \) value produced by GS with a given scheduling window, compared to that produced by LTF, is a direct measure of how effectively are IDEA-1 and IDEA-2 supported with that scheduling window. This is the consideration at the basis of the adaptive tuning of \( SW^* \) presented below. Before entering the details of the adaptive tuning, it is important to remark that the average waste of CPU time which should be reduced by GS, compared to LTF, is the average waste of CPU time on all the processors. This is because the reduction of \( WT \) on a single processor does not necessarily lead to performance improvements. For this reason, adaptiveness must be such that the same value for \( SW^* \) has to be adopted on all the processors in order to monitor the real impact of its variations on the performance.

We consider the execution of the simulation as divided into observation periods. Each period consists of \( N \) committed/rolled-back events. The value of \( N \) should be chosen in the way to ensure statistical data on \( WT \) collected in any period to be meaningful (suggestions to solve this problem have been already pointed out in other studies [16]). Furthermore, we assume there exists a master processor MP which gathers statistical data related to \( WT \) collected on the remote processors and adaptively computes the new value for \( SW^* \). This value is then notified to the slave processors. The interaction between MP and the slave processors is as follows. When the observation period expires at MP, it sends a statistic_request message to the slave processors. Upon the receipt of this message, any slave processor replies with a collected_statistic message whose payload is the locally observed \( WT \) value since the last request sent by MP was processed. When all the replies are received by MP, it computes the observed \( WT \) value of the simulation in the current observation period as the average of the \( WT \) values on all the processors.\(^4\)

\[^4\]The quantity \( 1 - P_r \) is widely known as the efficiency of the simulation.
It then selects the new value for \( SW^* \) and notifies it to the slave processors through an \( SW^* \) notification message. The new value is adopted for the next observation period. Note that the simulation is not frozen while the interaction between MP and the slave processors takes place.

The adaptive selection of the value of \( SW^* \) performed by MP is as follows. At the beginning of the simulation execution the value \( SW^* = 0 \) is selected and notified to all the slave processors. This means that during the first period GS behaves like LTF. At the end of the successive observation periods, the value of \( SW^* \) is increased by a quantity \( \delta \) if the observed \( WT \) of the simulation did not increase. If later quantity increases, then \( SW^* \) is decreased by \( 2\delta \) (in the case \( SW^* \) is less than \( 2\delta \), it is set to zero). The step for the increase is \( \delta \), the step for the decrease is \( 2\delta \); this difference is in order to quickly move the adequate scheduling window \( SW^* \) towards the value zero (i.e. to quickly move the behavior of GS towards the one of LTF) when successive increments of the value of \( WT \) are observed. This adaptive solution aims at keeping the value of \( SW^* \) as large as possible while still guaranteeing the feasibility of IDEA-1 (i.e. while still producing no relevant increase in the amount of rollback, expressed by \( P_r \), as compared to LTF scheduling). This is done in order to increase the expected cardinality of the good candidates set \( GC \) at the time the scheduling decision must be taken. This should actually lead to high flexibility of the scheduling decision as a function of the granularity of the events in \( GC \), thus producing throttling on very large grain ones, which is the final outcome of IDEA-2. If too large values for \( SW^* \) are selected then IDEA-1 could become infeasible thus originating an increase of \( P_r \) (i.e. an increase in the amount of rollback). If such an increase is poor and is still balanced by the gain from the throttling effect on large grain events, then \( SW^* \) is further increased. Otherwise, it is decreased.

Concerning the selection of the value for \( \delta \), preliminary performance results have shown that low values for \( \delta \), compared to the average timestamp increment of the simulation, usually produce the best performance results. As a general rule for the selection of \( \delta \) we suggest the following expression:

\[
\delta = \frac{\bar{T}}{10}
\]

where \( \bar{T} \) represents the average timestamp increment of the simulation.

Finally, we note that, from a “philosophical” point of view, our approach for the adaptive tuning of \( SW^* \) borrows from adaptive techniques for choosing the checkpointing protocol parameters in Time Warp simulations, for example the frequency of periodic checkpoints, such as the ones in [3, 11, 12].

### 3.4 Relation with Classical Throttling Algorithms

A number of throttling algorithms (e.g. [1, 2, 19]) have been proposed to associate with a non-executed event \( e \) an adequate execution delay in order to get good balance between the gain from the reduction of the probability for \( e \) to be eventually rolled back and the loss from the limitation of the optimism of the execution. Typically in these algorithms the delay is computed once the event \( e \) has been already scheduled for the execution, therefore the throttling is actuated at a higher level compared to that of the scheduling algorithm. This makes these algorithms different from our solution where the throttling on large grain events is actuated exactly at the level of the scheduling algorithm.

Although we have no empirical evidence yet, we argue that these two distinct types of throttling, being actuated at distinct levels, could be used in conjunction in order to further improve the performance.

### 4 Performance Data

In this section experimental results are reported to compare the performance achievable by using the GS scheduling algorithm to the one of the classical LTF algorithm. Before showing the results of the comparative analysis, we describe the main features of the used hardware/software architecture, present the selected benchmark and introduce the performance parameters we have measured.

As hardware architecture we used a cluster of machines (Pentium II 300 MHz - 128 Mbytes RAM) connected via fast switched Ethernet. The number of machines in the cluster is four. Inter-processor communication relies on message passing supported by PVM [20]. Our software is such that event cancellation is aggressive (i.e. antievents are sent as soon as the LP rolls back [6]) and fossil collection is executed periodically.

We tested the performance of the scheduling algorithms using the synthetic benchmark known as PHOLD model, originally presented in [5]. It consists of a fixed number of LPs and of a constant number of jobs circulating among the LPs (that is referred to as job population). Both the routing of jobs among the LPs and the timestamp increments are taken from some stochastic distributions. We have chosen this benchmark for two main reasons: (i) its parameters (e.g. event execution time and state saving cost) can be easily modified, (ii) it usually shows rollback behavior similar to many other synthetic benchmarks and to several real world models. In addition, it is important to remark that it is one of the most used benchmarks for testing the performance of both scheduling algorithms [14, 17] and checkpointing techniques [11, 13, 16, 18, 21].

The PHOLD model we considered is composed of 64 homogeneous LPs. Two different distributions were used for the timestamp increments: an exponential distribution with mean 1 simulated time unit and an uniform distribution in the interval [0,2] simulated time units. There are four hot spot LPs to which 30% of all the jobs are routed. The remaining percentage of jobs are equally likely to be routed to any LP. The spots randomly move among the LPs in the course of the simulation. In each simulation there are two distinct event types (i.e. two distinct job types), say light and heavy. Any light event takes about 200 microseconds to process and the effect of its execution is the production of a new light event. A heavy event takes a time which is about
10 times the one of a light event; its execution produces a new heavy event. This simulation model has resemblances to the model considered in [18] for testing the performance of a checkpointing technique specifically designed for simulations with high event granularity variance. It nicely approximates simulations of mobile communication systems. In this context, light events model position updates while heavy ones model resource allocation/utilization, which is typically more costly to simulate. Two distinct job populations were considered: one job per LP and 10 jobs per LP. In both cases 50% of all the jobs are light, 50% are heavy; they are randomly assigned to the LPs at the simulation starting. State saving is performed after the execution of each event; its cost has been fixed at about 15 microseconds.

The four configurations of the benchmark were run using all the 4 machines of the cluster. Each machine hosts the same number of LPs (no other user load runs on any machine). For the GS algorithm, the recalculation of the value of the adequate scheduling window \( SW^* \) is executed each 5000 events (i.e. the observation period at the master processor is fixed at 5000 committed/rolled-back events).

We report measures related to the following parameters:

- the amount of rollback (AR), that is, the ratio between the number of rolled back events and the total number of executed events (committed plus rolled back); this parameter corresponds exactly to the probability value \( P_c \) in expression (5); it indicates how good is a given scheduling algorithm from the point of view of the rollback originated during the simulation execution;

- the average granularity of the executed (committed plus rolled back) events (AG); this parameter corresponds exactly to the average granularity value \( \bar{G} \) defined in expression (6); it represents a metric for the granularity of the rolled back events; specifically, the larger the granularity of the rolled back events the larger the value of AG;

- the event rate (ER), that is, the number of committed events per second; this parameter indicates how fast is the simulation execution with a given scheduling algorithm, it is therefore representative of the achieved performance.

For the GS algorithm we report also the observed value of the average scheduling window length (ASW). For each configuration of the benchmark we report the average observed values of previous parameters, computed over 20 runs that were all done with different seeds for the random number generation. At least \( 2 \times 10^6 \) committed events were simulated in each run. The results for the case of exponential timestamp increment are reported in Table 1 and in Table 2. Those for the case of uniform distribution are reported in Table 3 and in Table 4. The values for AG are reported in microseconds, those for ASW in simulated time units.

The obtained data show that the amount of rollback AR under GS remains quite close to that obtained with LTF. The maximum difference is about 10%; it is observed for the case of exponential timestamp increment with a job population of 10 jobs per LP. This is achieved with values of ASW that in no case are much smaller (e.g. one order of magnitude or more) than the average timestamp increment of the simulation. Specifically, for the case of exponential timestamp increments, the observed values of ASW are in the order of 1/4, or 1/5 the average timestamp increment; for the case of uniform timestamp increments those values are in the order of 1/3, or 1/4 the average timestamp increment. This points out the feasibility of IDEA-1 since non-minimal scheduling window lengths can be actually used for increasing the flexibility of the scheduling decision as a function of the event granularity, without producing relevant negative effects from the point of view of the rollback originated in the simulation execution. We note that for both exponential and uniform timestamp increments, simulations with 1 job per LP show values of ASW higher than those with 10 jobs per LP. This behavior is an expected one when considering that large job populations originate high density of events in the simulated time. In this cases, the flexibility of the scheduling decision is actually increased even with small scheduling window lengths (i.e. the set of good candidate events \( G \bar{C} \) in (3) is likely to have large cardinality even with small scheduling windows).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AR</th>
<th>AG</th>
<th>ER</th>
<th>ASW</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTF</td>
<td>0.174</td>
<td>1109</td>
<td>1241</td>
<td>0.28</td>
</tr>
<tr>
<td>GS</td>
<td>0.177</td>
<td>1029</td>
<td>1309</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AR</th>
<th>AG</th>
<th>ER</th>
<th>ASW</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTF</td>
<td>0.111</td>
<td>1085</td>
<td>1359</td>
<td>0.19</td>
</tr>
<tr>
<td>GS</td>
<td>0.123</td>
<td>915</td>
<td>1637</td>
<td></td>
</tr>
</tbody>
</table>

The second point outlined by the results is a strong reduction of the values of AG when GS is adopted. Specifically, we get that the values of AG under GS are lower, between 6% and 16%, compared to those under LTF. This points out the real gain achievable through the throttling effect underlying IDEA-2, which strongly reduces the number of large grain events that are eventually rolled back.

When combining the results for AR and AG (i.e. when multiplying them, see expression (5)) we get that GS actually originates a reduction of the waste of CPU time between 4% and 10%. This allows GS to provide higher values for ER (between 3% and 8%), thus leading to faster execution of the simulation. The performance gain is achieved for all the four tested configurations, thus pointing out that GS could actually represent an adequate solution in simulations with high event granularity variance.

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 a
Time Warp simulation which is well suited for simulations with high variance of the event granularity. In our solution, the scheduling decision is a function of both timestamps and granularity values of the simulation events. The algorithm aims at reducing the average granularity of the rolled back events without increasing the whole amount of rollback of the simulation. The combination of these two features will possibly reduce the waste of CPU time due to rolled back events, thus yielding faster execution of the simulation.

We have tested the performance of the algorithm using a classical benchmark in different configurations. The obtained data point out that our solution allows faster simulation execution (up to 8%) in every considered configuration compared to the classical Lowest-Timestamp-First algorithm.

Acknowledgments. The authors thank the anonymous referees for helpful comments which have improved the content and the presentation of the paper. A special thank goes to Paolo Molinari for his help in the implementation of the simulation code.

References

Table 3. Unif. Timest. Incr. - 1 Job per LP

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AR</th>
<th>AG</th>
<th>ER</th>
<th>ASW</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTF</td>
<td>0.134</td>
<td>1012</td>
<td>1299</td>
<td>0.34</td>
</tr>
<tr>
<td>GS</td>
<td>0.135</td>
<td>1017</td>
<td>1299</td>
<td>0.34</td>
</tr>
</tbody>
</table>

Table 4. Unif. Timest. Incr. - 10 Jobs per LP

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AR</th>
<th>AG</th>
<th>ER</th>
<th>ASW</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTF</td>
<td>0.083</td>
<td>1123</td>
<td>1453</td>
<td>0.22</td>
</tr>
<tr>
<td>GS</td>
<td>0.091</td>
<td>949</td>
<td>1565</td>
<td>0.22</td>
</tr>
</tbody>
</table>