On the Processor Scheduling Problem in Time Warp Synchronization

FRANCESCO QUAGLIA
Università di Roma "La Sapienza"
and
VITTORIO CORTELLESSA
Università dell'Aquila

Time Warp is a synchronization mechanism for parallel/distributed simulation. It allows logical processes (LPs) to execute events without the guarantee of a causally consistent execution. Upon the detection of a causality violation, rollback procedures recover the state of the simulation to a correct value. When a rollback occurs there are two primary sources of performance loss: (1) CPU time must be spent for the execution of the rollback procedures and (2) waste of CPU time arises from the invalidation of event executions. In this paper we present a general framework for the problem of scheduling the next LP to be run on a processor in Time Warp simulations. The framework establishes a class of scheduling algorithms having the twofold aim to keep low the CPU time for the execution of the rollback procedures and also to guarantee low waste of time due to event executions invalidated by rollback. The combination of these two aims should actually lead to short completion time of Time Warp simulations.

We collocate existing scheduling algorithms within the framework, pointing out how they miss previous aims, at least partially. Then we instantiate a Window-based Grain Sensitive (WGS) scheduling algorithm relying on the framework, which pursues the above twofold aim. We also identify the proper conditions, associated with the simulation model execution, under which any algorithm exploiting the framework structure is expected to benefit the performance of the Time Warp mechanism. Empirical evidence from an experimental study of WGS on classical benchmarks and on a mobile communication system simulation fully confirms the theoretical outcomes.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems; D.4.1 [**Operating Systems**]: Process Management —*scheduling*; I.6.8 [**Simulation and Modeling**]: Types of Simulation — *discrete event*; *parallel*

General Terms: Theory, Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Optimistic synchronization

An earlier version of this paper by the same authors with title "Grain Sensitive Event Scheduling in Time Warp Parallel Discrete Event Simulation" appeared in *Proc. of the 14th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation (PADS'00).*

Authors' addresses: F. Quaglia, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy, email quaglia@dis.uniroma1.it; V. Cortellessa, Dipartimento di Informatica, Università dell'Aquila, Via Vetoio, Coppito, 67010 L'Aquila, Italy, email cortelle@di.univaq.it

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 [Fujimoto 1990a]. The execution of any simulation event at an LP usually modifies its state and possibly produces new events. Typically, the notification of a new event between distinct LPs consists of sending a message carrying the content and the occurrence time, namely *timestamp*, of the event.

The LPs must synchronize to ensure a timestamp ordered execution of events at each LP, which is a sufficient condition for the correctness of simulation results [Fujimoto 1990a]. In Time Warp synchronization [Jefferson 1985], 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. This is done by sending "antievents" that catch and annihilate the corresponding events. The event cancellation possibly leads other LPs to rollback. Specifically, if an LP receives an antievent corresponding to an already executed event, it must rollback as well.

A key problem to tackle in order to guarantee Time Warp synchronization being consistently efficient is how to schedule LPs for the execution on any processor. We recall that the need for a processor 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 multiple LPs that may simultaneously have at least one non-executed event in their event lists.

In the classical perspective, the term "good" for a scheduling algorithm has been 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 implies infrequent rollbacks and also short rollback length. Examples of algorithms pursuing this objective are in [Lin and Lazowska 1991; Quaglia 2000; Ronngren and Ayani 1994b; Som and Sargent 1998]. We argue that this perspective can be refined. Our intuition comes out from that the amount of rollback could only partially represent the final performance perceivable since it is not an exact measure of the real waste of CPU time associated with event executions invalidated by rollback. Hence, it is not an exact measure of the real rollback penalty. Specifically, many real world simulations, such as battlefields simulations or simulations of mobile communication systems, typically exhibit large variance of the event granularity (i.e. large variance of the event execution time). In this context, if the scheduling algorithm produces low amount of rollback but most of the invalidated events have large granularity we could still get large waste of CPU time, which possibly leads to poor performance.

Starting from this perspective, we present in this paper a general framework establishing a class of scheduling algorithms with the aim at ensuring:

Objective 1. Low amount of rollback; and also

Objective 2. That most of the invalidated event executions took short CPU time.

The combination of these two objectives can actually lead to fast completion of the simulation.

We show how existing scheduling algorithms, except the ones in [Palaniswamy and Wilsey 1994; Preiss et al. 1994], can be seen as particular instances of the class of algorithms associated with the framework. However, due to extreme behavior, these algorithms miss **Objective 2**. We also discuss the reason why the algorithms in [Palaniswamy and Wilsey 1994; Preiss et al. 1994] have no collocation within the framework. In the discussion we point out how these algorithms miss some control on **Objective 1**.

Starting from the framework we instantiate a Window-based Grain Sensitive scheduling algorithm (WGS) characterized by no extreme behavior. This allows WGS to pursue both previous objectives. In WGS the scheduling decision passes through two distinct steps. The first uses temporal information, namely timestamp values, to perform a filtering action for determining which LPs must be considered in the final scheduling decision. The second step (i.e. the final decision) relies on information on the (expected) amount of CPU time required by the events for their execution. Each step directly pursues one of the two previously mentioned objectives. As we will show, the first step of the scheduling decision makes use of a parameter whose value can be tuned in accordance with the (dynamic) characteristics of simulated applications. This yields WGS to be adequate for any application setting.

By a discussion we identify under which conditions, proper of the simulation model execution, any algorithm resulting as an instance of the algorithm class associated with the framework is expected to provide performance improvements. The results of an empirical study of WGS on classical benchmarks in several different configurations and on a mobile communication system simulation actually confirm the discussion outcomes.

The remainder of the paper is organized as follows. In Section 2 we present the framework. In Section 3 we report an overview of existing scheduling algorithms and collocate these algorithms within the framework. The WGS algorithm is instantiated in Section 4. The discussion on the relations between effectiveness and conditions proper of the simulation model execution is presented in Section 5. The results of the empirical study are reported in Section 6 and in Section 7. A somehow hidden effect of the WGS algorithm on the Global Virtual Time advancement is discussed in Section 8. Assessments and conclusion are reported in Section 9.

2. A FRAMEWORK FOR THE SCHEDULING PROBLEM

Figure 1 shows the framework structure we propose for establishing the class of scheduling algorithms pursuing both **Objective 1** and **Objective 2**. Each rectangular box represents a system associated with a specific functionality, whereas each oval box represents a set of events. We shall proceed explaining the framework structure into details:

Filtering. We call the Candidates set (C) the set containing, for every LP hosted by the processor, the next to be executed event, if any, i.e. the one with the lowest

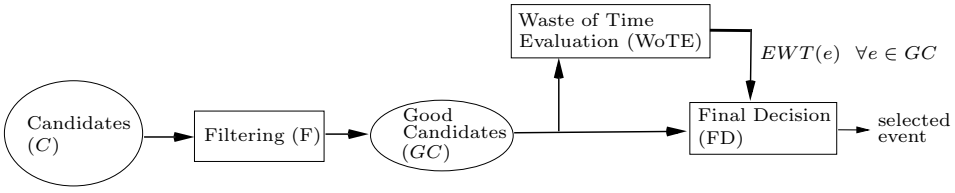

Fig. 1. General framework for the scheduling problem.

timestamp among the events already incorporated into the event list of that LP. In our framework we envisage the existence of a Filtering system (F) which identifies a subset of C, namely the Good Candidates set (GC), such that the selection for execution of any event belonging to GC contributes to keep low the amount of rollback of the simulation. The LPs associated with the events in GC are therefore good candidates for the scheduling according to **Objective 1**. Hence the F system aims at directly pursuing this objective.

It is important to remark that rollback derives from optimism in the event execution which, in turn, can be seen as the conjunction of two distinct properties known as *aggressiveness* and *risk* [Reynolds, Jr. 1988]. Aggressiveness is the property by which LPs are allowed to execute their events without the guarantee of no timestamp order violation. Risk is the property by which new events produced by aggressive event execution are propagated in the system. When an aggressive event execution is invalidated, possibly antievents are sent to catch and annihilate the events produced and propagated as a result of that execution, which might lead other LPs to roll back. This points out how both aggressiveness and risk may contribute to rollback.

The ideal behavior for system F would be to determine the set GC on the basis of the likelihood of an aggressive execution to be eventually invalidated and also on the basis of the risk associated with the execution and its real impact on rollback. However, as we will show while collocating existing algorithms within the framework, it is common to attempt to keep low the amount of rollback by bounding primarily the effects of aggressiveness. This design choice typically derives from the fact that the effects of risk on rollback arise conditional to aggressiveness ultimately resulting in some timestamp order violation. Therefore, reducing the likelihood of violations due to aggressiveness is expected to automatically bound the effects of risk. In other words, aggressiveness is commonly recognized as the primary property to take into account.

Note that this viewpoint is not exclusive of designers of processor scheduling algorithms, but is common also to designers of synchronization mechanisms with controlled (limited) optimism. Specifically, for several of these mechanisms (see, for example, [Ferscha 1995; Ferscha and Luthi 1995; Hamnes and Tripathi 1994; Mascarenhas et al. 1998; Srinivasan and Reynolds, Jr. 1998]) optimism control actually means direct limitation on aggressiveness only.

Waste of Time Evaluation. We also envisage the presence of a Waste of Time Evaluation system (WoTE) which computes for every event $e \in GC$ the Expected Waste of Time $EWT(e)$ associated with the selection of the event e for the execution.

The waste of time associated with the invalidation of the execution of the event e consists of:

(1) execution time for the code instructions associated with the event e;
(2) notification time for the events (produced by e's execution) that must be cancelled.

Waste of time in point (1) relates to aggressiveness in the event execution, whereas waste of time in point (2) relates to risk associated with the event execution. This points out how, like for the case of the F system, also for WoTE the ideal behavior would be to determine EWT values on the basis of the contributions associated with both aggressiveness and risk.

However we note that taking into account the contributions associated with aggressiveness and discarding those associated with risk should often represent an adequate solution in practice. This is due to the fact that parallel simulation is typically used for models with medium/large event granularity (in the order of several tens up to hundreds of microseconds on current conventional architectures) that is usually predominant as compared to the CPU overhead for notification of new events, which is typically in the order of few microseconds. Specifically, parallel discrete event simulation is usually supported by shared memory multiprocessors and distributed memory systems coupled with very light communication layers that typically do not involve operating system functionalities (see for example [Pakin et al. 1995]). Therefore, regardless of notification taking place between LPs hosted by remote processors or not, notification time typically involves only copying the message content into/from a proper buffer. This operation is not relevantly time consuming ([1]).

Final Decision. Once computed EWT values, a Final Decision system (FD) uses them to finally determine the event that has to be selected for execution, and therefore the LP that has to be scheduled. FD selects the event $e^* \in GC$ associated with the minimum EWT value. The execution of e^* should keep low the amount of rollback of the simulation (**Objective 1**), since e^* has been selected as a good candidate by the filtering system F and, in addition, should produce the following positive effect on performance: the expected waste of time associated with the scheduling decision is the minimum predictable one. This directly contributes to **Objective 2**.

The instantiation of any algorithm deriving from this framework passes therefore through two distinct phases:

- introducing a filtering system F for determining GC;
- defining an estimation system WoTE for determining the EWT values associated with the events in GC.

There are different design alternatives for each phase, therefore different scheduling algorithms may be devised by changing F and WoTE.

[1]For the case of a distributed memory system, the real difference between local and remote notification is in the delivery delay of the notification message since remote notification requires data transfer through the network hardware. However, this transfer does not impose additional CPU time penalties at the application level.

2.1 Implementation Issues

The framework relies on three distinct blocks, namely F, WoTE and FD. At a logical level, each block performs a distinct action and the interaction between blocks takes place as shown by arrows in the framework structure (see Figure 1). Specifically F provides the set GC to both WoTE and FD, and WoTE provides EWT values to FD. However, when instantiating, and then implementing, a specific scheduling algorithm, there exists the possibility that work performed by one block to produce its output can be used to simplify the work of another block. Consider for example a scheduling algorithm relying on a system F that, in order to produce its output, needs to estimate the probabilities for the events in C to be eventually rolled back if currently selected for the execution. These estimated probability values could be used by the WoTE system to compute EWT values without the need for a new estimation. The advantage of re-using estimated probability values consists of a reduction of the overhead associated with functionalities of the two systems.

With respect to the reduction of the overhead, we remark that the EWT values in input to FD could be (partially) left as symbolic expressions by WoTE provided that FD has the capability to identify the event to be selected without the need for estimating parameters to solve the symbolic expressions. From the experience done while designing the WGS algorithm we shall present in Section 4, the structure of the F system has an impact on this capability. Specifically, in the WGS algorithm, WoTE is not required to explicitly estimate for any event $e \in GC$ the probability of being rolled back, if executed, even though this probability is a parameter of the EWT expression. This is due to the particular structure of the F system associated with WGS, whose filtering action produces a set GC containing events expected to have "almost" the same probability to be eventually rolled back, if selected for the execution. Thus, the retrieving of the event in GC associated with the minimum EWT value can take place with no explicit estimation of these probability values.

Prior to presenting the WGS algorithm, let us survey existing algorithms and let us discuss their relation with the structure of the framework.

3. EXISTING SCHEDULING ALGORITHMS

3.1 Overview

The standard solution for the scheduling problem in Time Warp is the Lowest-Timestamp-First algorithm (LTF) [Lin and Lazowska 1991] which always schedules as next LP to be run on a processor the one associated with the non-executed event having the minimum timestamp. LTF implicitly assumes that the event with the minimum timestamp has the lowest probability to be eventually rolled back. Motivations for this presumption will be discussed in Section 4 since they constitute the basis for the WGS algorithm we propose. If this presumption reveals true, as typically occurs [Preiss et al. 1992], then LTF will keep low the amount of rollback by keeping low both its frequency and its length.

Another scheduling algorithm, namely Lowest-Local-Virtual-Time-First (LLVTF) [Preiss et al. 1994], gives higher priority to LPs having lower simulation clocks (i.e. lower LVTs). In particular, LLVTF chooses for running the LP with the lowest LVT value and having at least one non-executed event in its event list. 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 simulation 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. An expression for the useful work is introduced as a refinement of the classical frequency of committed events evaluated overall the CPU time used by the LP. Higher priority is assigned to LPs having higher values of their useful work.

A rather different solution, namely Service Oriented scheduling (SO), is presented in [Ronngren and Ayani 1994b]. 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 promptly deliver those events to the recipient LPs, thus reducing the probability of timestamp order violations on these LPs. Such an approach needs the capability of the LPs to predict the timestamps of events not yet 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 LP scheduled for running is the one associated with the event having the minimum real probability to be rolled back in the future. 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 not to be eventually rolled back if currently executed. The LP associated with the event having the highest estimated probability value is scheduled for the execution. This solution will produce low amount of rollback each time the estimated probability values are good approximations of the real ones. This should happen in simulations with regular patterns for the arrival process of the events in simulation time.

Finally, in [Quaglia 2000] a State Based scheduling algorithm (SB) has been presented. In this algorithm, the scheduling priority of any LP is computed on the basis of state information related to the LPs in its immediate predecessor set. Specifically, higher priority is assigned to the LPs, if any, whose next event may be rolled back only conditional a rollback occurs on an LP in their immediate predecessor set. This possibly keeps low the amount of rollback. If none of previous LPs is detected at the scheduling time, then SB acts as the classical LTF.

3.2 Collocation within the Framework

We can think of most existing scheduling algorithms as particular instances of the algorithm class associated with the framework we propose. They can be seen as instances characterized by extreme behaviors due to the particular nature of their F systems.

Specifically, we can think of LTF, SO, P and SB as scheduling algorithms in which the system F has the following property: if C is not empty, it always originates a set GC that contains a single event. More precisely:

—LTF can be though as of an algorithm where F originates a set GC that contains the event belonging to C having the minimum timestamp;

—SO can be thought as of an algorithm where F predicts the timestamp of future

events resulting from the execution of events in C and then originates a set GC containing the event belonging to C which will produce the event with the minimum predicted timestamp;

—P can be thought as of an algorithm where F predicts the probability for the events in C not to be eventually rolled back and then originates a set GC containing only the event belonging to C associated with the highest predicted probability value;

—SB can be thought as of an algorithm where F collects data related to the immediate predecessor sets of the LPs; then, starting from C originates a set GC containing either one event that can be rolled back conditional a rollback occurs elsewhere in the system or the event with the minimum timestamp.

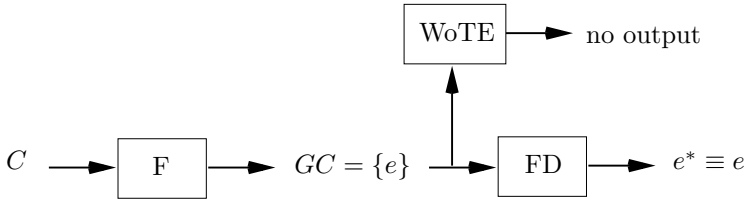

Fig. 2. Specialization of the framework.

In all these algorithms there is actually no need for output values provided by a WoTE system as the final decision system FD will always select for execution the unique event belonging to GC, independently of any EWT value associated with it. Basically the framework structure can be specialized for these algorithms as shown in Figure 2, where the WoTE system produces no output and the FD system has a trivial structure since it takes as input the set $GC = \{e\}$ and directly transfers the event e to the output.

These algorithms can be seen as "greedy" solutions that try to optimize the scheduling decision exclusively through actions associated with the F system. They pursue only **Objective 1**, therefore they miss a kind of control on the waste of CPU time due to event execution invalidation, namely **Objective 2**. This could be penalizing depending on proper features of the application. As pointed out in the Introduction, if we consider simulation applications with large variance of event granularity (recall that several real world simulation applications exhibit this feature), then keeping low the amount of rollback might not be a relevant objective as long as it is not coupled with the objective to guarantee that the majority of the rolled back events are fine grain ones. If this coupling is not achieved, then there is the risk of relatively large waste of CPU time even with low amount of rollback.

Finally, we note that most of these algorithms (LTF, P and SB) have an F system which has no direct control on how the risk proper of Time Warp synchronization actually affects the amount of rollback. In these solutions, the F system identifies as good candidate event the one whose aggressive execution is considered as less likely to correspond to a timestamp order violation. In other words these algorithms select for execution the event that (based on some criterion) is considered as less likely to be eventually rolled back. This happens independently of the risk associated with

the new events produced by the execution. Instead, the SO algorithm includes, at some extent, the notion of risk at the level of the F system since the event to be executed is selected on the basis of features, namely predicted timestamp values, associated with the new events that will be produced by the execution.

The algorithms LLVTF and AC have no direct collocation within the framework. This is because the framework establishes a class of algorithms that we can define as "event-based". Specifically, the philosophy underling these algorithms is as follows: first consider the events in C and extract one of them (passing through F and FD), as a consequence schedule for running the associated LP. On the contrary, LLVTF and AC can be seen as "LP-based" solutions relying on the philosophy: schedule for running a given LP, as a consequence extract from C the associated event. This approach does not fully capture the likelihood of the event extracted from C to be eventually rolled back. For instance, consider a case with two LPs, namely LP_a and LP_b hosted by the same processor, having Local Virtual Times LVT_a and LVT_b, and having next events with timestamps 20 and 100 respectively. If $LVT_a > LVT_b$ then LLVTF will favor the scheduling of LP_b even though its next event is extremely likely to have higher probability to be eventually rolled back being much far ahead in simulation time. This behavior leads LLVTF to miss a kind of control on the amount of rollback, that is on **Objective 1**. The same argument can be reported for the case of AC. Specifically, if we refer to the previous example and suppose LP_a and LP_b have useful work values uw_a and uw_b, respectively, with $uw_a < uw_b$, then AC will favor the scheduling of LP_b (just like LLVTF does) even though, as stated before, its next event is more likely to be eventually rolled back. Therefore, also AC misses a kind of control on **Objective 1**.

4. INSTANTIATION OF THE WGS ALGORITHM

To present the WGS algorithm we will proceed along the following line. We first design the F system. Then we present the WoTE and the FD systems to be coupled with F. Then at the end of the section a remark on the effects of WGS is reported.

4.1 Design of the F System

The design of the F system relies on the extension of basic considerations underlying the classical LTF scheduling algorithm. As discussed in Section 3, LTF always selects for the execution the event in C with the minimum timestamp. (If several events in C have the minimum timestamp, then LTF randomly selects among them the event to be executed.) We will refer to the event selected by LTF as $min(C)$. The advantages of this choice are as follows:

A. Selecting $min(C)$ will never result in any timestamp order violation if no messages carrying events/antievents will ever be delivered to local LPs in the course of the simulation execution. In other words, violations may only be due to interprocessor communication.

B. Although no validated model has been developed for this, it is widely recognized that the difference between the timestamp of a non-executed event e, that we will refer to as $ts(e)$, and the current Global Virtual Time (GVT) of the simulation is representative of the likelihood for e to be eventually rolled back if currently exe-

cuted ([2]). More precisely, the less the value of the difference $ts(e) - GVT$, the lower the probability that e will be rolled back in the future of the simulation execution. This is because a timestamp value $ts(e)$ close to GVT typically means low probability that in-transit or future messages will eventually carry events/antievents with timestamp values less than $ts(e)$, due to the low width of the simulation time interval they should fall in.

Among the events in C, $min(C)$ has the minimum timestamp distance from GVT, therefore, it is the best event to select in respect to previous presumption.

System F underlying WGS is based on the idea of identifying a set of events belonging to C such that the selection of any of them for execution preserves the advantages associated with the selection of $min(C)$ (i.e. the advantages in points A. and B.). Specifically, the design of system F relies on the following consideration: if the timestamp of an event $e \in C$ distinct from $min(C)$ is "very close" to the timestamp of $min(C)$, i.e. the two events are in narrow time proximity, then:

(i) Since all the events in C belong to distinct LPs, narrow proximity between $min(C)$ and e in simulation time is an indicator that there is high likelihood that no causal relation exists between any event in C and e. (With causal relation we intend the classical perspective according to which the execution of any event in C distinct from e will eventually cause an event e' to be produced, with $ts(e') < ts(e)$, destined to the LP that must execute e.)
Therefore, selecting e for execution is likely not to originate timestamp order violations if no interprocessor communication occurs. As a result, the advantage described in point A. is likely to be preserved.

(ii) $min(C)$ and e are likely to have about the same probability to be eventually rolled back due to interprocessor communication since their timestamps have about the same distance from the current GVT of the simulation. Therefore, selecting e for execution is likely not to increase the probability of rollback occurrence due to interprocessor communication. As a result, the advantage described in point B. is likely to be preserved.

Generalizing the previous consideration, we get that there may exist an Adequate Scheduling Window, denoted as ASW, determining the following simulation time interval $I(ASW)$:

$$I(ASW) = [ts(min(C)), ts(min(C)) + ASW] \tag{1}$$

such that the below Interval Property ($\mathcal{IP}$) holds:

Interval Property - ($\mathcal{IP}$)
The selection for execution of any event $e \in C$ distinct from $min(C)$ such that

[2]We recall that the GVT of a Time Warp simulation is defined as the minimum of the timestamps of events not yet executed that have been already incorporated into the event lists of the recipient LPs and the timestamps of events/antievents carried by messages still in transit. GVT represents the *commitment horizon* of the simulation as no LP will ever rollback to a simulation time before GVT, i.e. no event/antievent with timestamp lower than the current GVT value will ever be delivered to any LP.

$ts(e) \in I(ASW)$ has the same advantages, in terms of attempt to keep low the amount of rollback, as the selection of $min(C)$.

In the perspective of the design of an F system, all the events belonging to C and having timestamp in the interval $I(ASW)$ are good candidates for the execution, because any of them is likely not to increase the amount of rollback of the simulation as compared to that resulting from LTF scheduling. (Note that we are implicitly considering LTF as a reference point with respect to **Objective 1** for the design of the F system underlying WGS. The reason for this choice is that LTF is the solution that generally ensures the lowest amount of rollback.) Therefore, we can think of a first design of system F as follows:

$$\mathrm{F}: \quad GC = \{e \in C \mid ts(e) \in I(ASW)\} \tag{2}$$

Note that LTF has the property that event selection does not take into account the risk associated with event execution. Instead, the only relevant parameter is the likelihood for the aggressiveness of the execution to ultimately result in a timestamp order violation. System F in (2), being derived from basic motivations underlying LTF, inherits this property.

We observe that there may exist an interval of values for ASW (with the value zero as a left extreme) which ensure $\mathcal{IP}$. By construction, the effectiveness of the filtering action performed by F is strongly dependent on which value in the interval is selected for ASW. Let us discuss this point. If ASW is too small then upon the filtering action the set GC might contain very few events (or at worst a single event). In this case there could be excessive greediness at the level of the F system which, similarly to the algorithms in [Lin and Lazowska 1991; Quaglia 2000; Ronngren and Ayani 1994b; Som and Sargent 1998], could prevent the FD system from effectively contributing in pursuing **Objective 2** on the basis of EWT values predicted by WoTE. In other words, if $I(ASW)$ is too narrow, then some events belonging to C, whose execution is expected to produce the same advantages as the execution of $min(C)$ in terms of amount of rollback, could be excluded by the set GC. Therefore, these events will not be considered in the final decision performed by FD even if they have very low EWT values. As a consequence the final scheduling decision could result as poorly optimized. To overcome this drawback, the F system should filter events using the Maximum Adequate Scheduling Window, denoted as $MASW$, that is the maximum ASW value still ensuring $\mathcal{IP}$. Therefore, the filtering action performed by F should be as follows:

$$\mathrm{F}: \quad GC = \{e \in C \mid ts(e) \in I(MASW)\} \tag{3}$$

This refined structure of F allows maximal cardinality of the set GC while still guaranteeing low amount of rollback. Figure 3 shows how the size of the window may actually affect the cardinality of GC. Issues related to the identification of the value of $MASW$ for any specific simulation application will be discussed in Section 6.5.

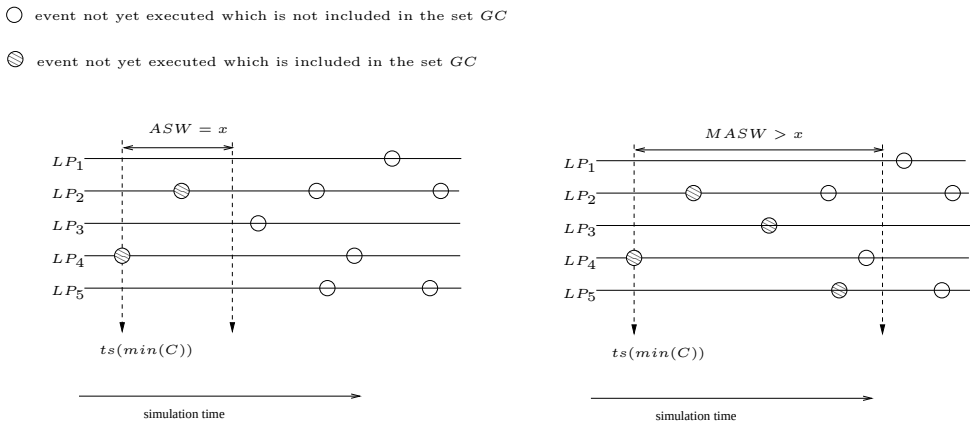

Fig. 3. Cardinality of GC vs the size of the Adequate Scheduling Window.

4.2 Design of WoTE and FD Systems

In the design of system WoTE we decide to neglect effects associated with risk on the waste of time, and to focus only on waste of time due to aggressiveness. Therefore, the expected waste of time model that we adopt is a simplification of an ideal one. Motivations for this simplification have already been discussed in Section 2, together with the general description of the functionalities of the WoTE system.

To present the model let us introduce few notations:

$P(e)$ denotes the probability for the event $e \in GC$ to be eventually rolled back if selected for the execution;

$G(e)$ denotes the expected granularity of the event e, that is the expected CPU time for the code instructions associated with the event e (it does not include the CPU time for the notification of new events possibly produced during the execution).

By previous notations and by the above design choice, the model we assume for $EWT(e)$ is as follows:

$$EWT(e) = P(e)G(e) \tag{4}$$

Therefore, the task of the WoTE system is to compute for any event $e \in GC$ the value of $EWT(e)$ as expressed by (4). In the general case this involves the estimation of $P(e)$ and, in case of unknown expected event granularity, it involves also the estimation of $G(e)$. However, due to the particular structure of system F previously presented, we can design an FD system that has the capability to produce its output while leaving $P(e)$ as a symbolic expression. This has the advantage of lightening the task of WoTE. (As an extreme, if the expected granularity values of the events of any type occurring at any LP are known in advance to the simulation execution, then no computation at all is to be performed by WoTE.)

Specifically, F originates the set GC on the basis of the $\mathcal{IP}$ property. This property, by its nature, for any event $e \in GC$ distinct from $min(C)$ actually enables

the probability $P(e)$ to be approximated by $P(min(C))$. Hence, for any event $e \in GC$ we can approximate EWT as:

$$EWT(e) = P(min(C))G(e) \tag{5}$$

Using the approximation based model in (5) to determine the input values to FD, the identification of the event $e^* \in GC$ associated with the minimum EWT value boils down to the identification of the event $e^* \in GC$ associated with the minimum expected granularity value. Note that there might be multiple events in GC with minimum expected granularity, so we let FD select, among these events, the one with the minimum timestamp (if ties still exist, they are broken randomly). This is a design choice to adhere to a kind of LTF policy among these events. Hence, the final structure of the FD system is as follows:

$$\begin{aligned} \text{FD}: \quad & S = \{e | (e \in GC) \wedge (\forall e' \in GC \; G(e) \leq G(e'))\} \\ & e^* \equiv e \in S: \; \forall e' \in S \;\; ts(e) \leq ts(e') \end{aligned} \tag{6}$$

With the structure of FD in (6), WoTE is only accounted to perform the estimate of expected granularity values, if really required. This could be implemented in several different ways. A simple solution consists of estimating the expected granularity value of an event of type τ occurring at the x-th LP using the arithmetic mean computed over n_τ^x observations. Denoting with $\delta_{\tau,j}^x$ the j-th observed amount of CPU time for an event of type τ occurring at the x-th LP ([3]), then the WoTE system can be expressed as:

$$\begin{aligned} \text{WoTE}: \quad \forall e \in GC: \quad & \text{if } (type(e) = \tau) \wedge (e \text{ to be executed by the } x\text{-th LP}) \\ & \text{then } G(e) = \frac{\sum_{j=1}^{n_\tau^x} \delta_{\tau,j}^x}{n_\tau^x} \end{aligned} \tag{7}$$

Adopting this solution has also the advantage that the update of the estimate can be done incrementally. More precisely, indicating with $G_h(e)$ the estimated value computed by using the first h collected observations (i.e. $G_h(e) = 1/h \sum_{j=1}^{h} \delta_{\tau,j}^x$), upon the collection of the $h+1$-th observation, the estimate is updated efficiently as:

$$G_{h+1}(e) = \frac{hG_h(e) + \delta_{\tau,h+1}^x}{h+1} \tag{8}$$

Different, more sophisticated statistical methods could be adopted for the implementation of the functionality of WoTE. However, we recall that changing the estimation method adopted for the implementation of WoTE has no effect on the structure of the systems F and FD we have presented.

[3]In case $\delta_{\tau,j}^x$ is not deterministic or is not known in advance of the execution, solutions to measure it at the price of negligible overhead have been recently discussed in [Quaglia 2001].

4.3 A Remark on the Effects of WGS

As pointed out in Section 4.1, in the common belief the difference between the timestamp $t(e)$ of a non-executed event e and the current GVT value, namely the commitment horizon, is recognized as being representative of the likelihood for e to be eventually rolled back if currently executed. Specifically, likelihood of rollback is expected to increase with increase in the distance from the GVT. Although WGS might not select for execution the event $min(C)$, it does not contrast such a common belief since it includes in the set GC only those events whose timestamps have adequately short distances from the GVT, and for which the $\mathcal{IP}$ property holds. Then, among those events, WGS favors the execution of the event with lightest expected granularity, even if it has not the minimum timestamp. The direct consequence is that the execution of a coarse grain event $e \in GC$, if any, is delayed. Such a delay, combined with the monotonic increase of the GVT is expected to produce a reduction of the likelihood for e to be eventually rolled back (if eventually executed)[4] [Ferscha 1995; Ferscha and Luthi 1995; Srinivasan and Reynolds, Jr. 1998]. Given that e is a costly event, this might relevantly reduce the waste of time. In other words, delaying the execution of a costly event (even if it has the minimum timestamp among the events in the set C) allows us to reconsider that same event in the scheduling process at a later time, when higher likelihood for definitive commitment of the event execution is expected.

5. BACK TO THE FRAMEWORK

It is widely recognized that the real rollback pattern, namely rollback frequency and rollback length, associated with a given simulation model execution depends on the adopted processor scheduling algorithm as well as on a number of other features characterizing both the simulation model and the execution itself. Examples of these features are the communication graph of the LPs, the number of used processors and how the LPs are mapped onto the processors, just to name some.

As pointed out in the Introduction, a good scheduling algorithm is classically identified as an algorithm having the capability to maintain the amount of rollback as low as possible, compatibly to those features associated with the simulation model and its execution. The framework defines a class of good algorithms since they pursue **Objective 1** through the filtering action performed at the level of system F. If the filtering action, combined with those features associated with the simulation model and its execution, will ultimately result in a minimal amount of rollback (i.e. a minimal amount of rolled back events), then pursuing **Objective 2** in cascade with **Objective 1** is expected to yield a negligible additional performance improvement since waste of time due to rolled back events represents a negligible percentage of the completion time of the parallel simulation execution.

Instead, consider the situation in which the out-coming amount of rollback for the specific execution is non-minimal. In this case, pursuing **Objective 2** in cascade with **Objective 1** might yield a consistent performance improvement, since waste of time due to rolled back events does not represent a negligible percentage of the completion time of the parallel simulation execution.

[4]This is also knows as *event throttling*.

5.1 A Deeper Look at **Objective 2**

In this section we show that, even in case of non-minimal amount of rollback, pursuing **Objective 2** in cascade with **Objective 1** could fail to produce a relevant reduction of the amount of waste of time, depending on proper dynamics that characterize the simulation model execution. This viewpoint is made clear through the following discussion.

Suppose processor p hosts a set of m LPs, namely $\{LP_1, \ldots, LP_m\}$. With each LP_j in the set we associate a path of executed events, potentially empty, that we refer to as $PATH_j$. Events executed by LP_j and then rolled back are not included in $PATH_j$. In other words, $PATH_j$ contains all the events occurred at LP_j with timestamp less than or equal to the current value of the local virtual time LVT_j of LP_j.

Some of the events in $PATH_j$ are committed events and they will never be rolled back. Instead, some other events in $PATH_j$, namely the more recently executed ones, might be uncommitted. Therefore they might be rolled back in the future of the simulation execution. Note that, at any given instant in real time, the composition of $PATH_j$ depends on proper dynamics of the simulation model execution, including the results of scheduling decisions taken on processor p up to that real time instant.

We denote as X_j the expected waste of time from $PATH_j$ at the real time instant the scheduling decision must be taken on processor p. In other words, X_j denotes the waste of time $PATH_j$ is expected to produce due the fact that $PATH_j$ itself might contain uncommitted events that might be eventually rolled back.

At the time the scheduling decision must be taken, the total expected waste of time on processor p, which we denote as X^T, can be evaluated as:

$$X^T = \Sigma_{j=1}^{m} X_j \tag{9}$$

Consider the set GC defined at the time the scheduling decision must be taken on the basis of the filtering action performed by system F. If GC contains an event to be executed by LP_j, then we denote that event as e_j. Also, suppose that:

$$\exists e_k \in GC : \forall e_i \in GC \ (e_i \neq e_k) \ EWT(e_k) < EWT(e_i) \tag{10}$$

In other words, suppose that at the time the scheduling decision must be taken, e_k is associated with the minimum expected waste of time value among the not yet executed events belonging to GC.

There are two possible scenarios:

Scenario A. **Objective 2** is effectively pursued by the scheduling algorithm so that the event e_k is selected for the execution on processor p. In this case we get a projection $X^{T'}$ for the total waste of time expectation on p which is equal to:

$$X^{T'} = X^T + EWT(e_k) \tag{11}$$

We have used the term "projection" since the real value of $X^{T'}$ after the execution of the event e_k might be different from the value in expression (11) due to the fact

that both X^T and $EWT(e_k)$ might change depending on current conditions after the execution of e_k.

Scenario B. For some reason, the scheduling algorithm decides to temporarily ignore **Objective 2** and selects for the execution an event $e_i \neq e_k$, with $EWT(e_i) > EWT(e_k)$. In other words, the algorithm does not take the "best suited" decision on the basis of expected waste of time projections relying on EWT values. In this case we get a projection $X^{T''}$ for the total waste of time expectation on p which is equal to:

$$X^{T''} = X^T + EWT(e_i) \tag{12}$$

Let us observe that if both inequalities $X^T >> EWT(e_k)$ and $X^T >> EWT(e_i)$ hold, then both projections $X^{T'}$ and $X^{T''}$ can be approximated by X^T. This indicates that, in case the additional waste of time due to any event selected for the execution is a negligible fraction of the total expected waste of time on p, then pursuing **Objective 2** (Scenario A) or not pursuing it (Scenario B) has no practical effect on the total waste of time expectation. In other words, adding e_k to $PATH_k$ or adding e_i to $PATH_i$ have, in practice, the same final effect in terms of expected waste of time.

There are two cases in which both previous inequalities are verified:

(i) The expected waste of time associated with the selected event (e_k or e_i) is much lower than some X_j values appearing in expression (9).

(ii) The expected waste of time associated with the selected event (e_k or e_i) is comparable with (or even higher than) any X_j value, but is negligible as compared to their sum, due to the fact that m is large (i.e. due to the fact that a large number of LPs are hosted by processor p).

Both cases (i) and (ii) are expressions of a similar phenomenon. In particular, in both cases we have that, given the expected waste of time projection we get by adding e_k to $PATH_k$, or by adding e_i to $PATH_i$, then the relevant part of waste of time sources in the projection is related to events that did not occur in the "immediate past of the execution". We name this phenomenon as *Aging of Relevant Waste of Time Sources* (ARWTS).

Specifically, case (i) denotes that there are some very large X_j values that represent a substantial contribution in the projection. This is an indication that a long rollback is expected to occur on some $PATH_j$'s, involving events related to a quite far past of the execution. On the other hand, in case (ii) any X_j contribution is not substantial. But, given that the LPs hosted by the processor are very numerous, most contributions will be associated with $PATH_j$'s having final segments constituted by events that have been executed quite far in the past (as the associated LPs have not been scheduled for the execution since a quite earlier time).

6. EMPIRICAL RESULTS WITH SYNTHETIC BENCHMARKS

In this section we present a two-part experimental analysis of WGS conducted using classical synthetic benchmarks. The two-part organization derives from issues raised in Section 5. Specifically, in the first part we test WGS against "favorable"

test cases, namely test cases where (i) the simulation model execution exhibits a non-minimal amount of rollback, therefore, as discussed in Section 5, pursuing **Objective 2** in cascade with **Objective 1** is relevant, and (ii) the execution is not affected by the ARWTS phenomenon depicted in Section 5.1. In the second part, we change features associated with the simulation model execution in order to originate "non-favorable" test cases that help in providing empirical validation of the ideas expressed in Section 5. Evaluation of WGS on a real world simulation application is delayed to Section 7.

Before entering the description of the test cases and of the associated results, we provide details on the testing environment and on the testing methodology.

6.1 Testing Environment

The experiments were all performed on a cluster of PCs (Pentium II 300 MHz - 128 MB RAM - 512 KB second level cache) running LINUX as operating system, interconnected by a high speed Myrinet switch. Any PC is connected to the Myrinet switch through an interface implemented on a card, namely the M2M-PCI32C card, consisting of a LANai processor (Version 4 [MYRICOM 1999]), a local memory and supports for DMA.

The LANai processor runs a control program that supports send and receive operations. This program can be designed according to the requirements of the specific application. We have developed a high speed layer, namely Minimal Fast Messages (MFM), which is tailored for optimizing the delivery delay of small size messages, typical of parallel discrete event simulation. Each message is transferred within a single packet whose size can be selected at compilation time in order to fit the requirements of the overlying application. Reliability of message delivery is ensured through an acknowledgment mechanism implemented at the level of the control program. The packet size we have used is 40 bytes (this fits the requirements of the selected test cases), therefore messages carrying any event/antievent and control messages are actually transferred through packets of the same size. For packet size 40 bytes, the delivery delay, intended as the time to transfer a message of up to 40 bytes between two application address spaces on distinct machines, is in the order of 20/25 microseconds when no congestion occurs on the switch.

With respect to other main features of the software being used, we remark that memory space for new entries into the event lists of the LPs is dynamically allocated by using classical `malloc()` calls, and the event lists are implemented as simple linked lists. Instead, pre-allocated memory has been used for the entries of the stack of saved state vectors. LPs are implemented as application level threads. Finally, the cancellation phase is implemented following the aggressive strategy, that is antievents are sent as soon as the LP rolls back [Gafni 1985].

6.2 Testing Methodology

A measure of success of any scheduling algorithm for Time Warp simulators is how significantly it accelerates the simulation model execution. Major sources for the acceleration are the reduction of the amount of rollback and, as suggested by the framework we propose, the reduction of the real waste of time associated with events that are eventually rolled back. In our empirical study, the acceleration provided by WGS is evaluated against the execution speed produced by the LTF algorithm.

Therefore, LTF is our reference algorithm in the analysis. This choice is determined by the fact that LTF is, in practice, the standard solution. The parameter we use to measure the execution speed, and thus the acceleration, is the *event rate*, namely the number of committed events per second.

Given that system F is defined by extending the ideas LTF relies on, we expect that the acceleration provided by WGS, if any, will not come out from a reduction of the amount of rollback as compared to LTF. Instead, we expect it to come out from a reduction of the real waste of time due to rolled back events. In order to provide empirical evidence of this, we report also measures related to the rollback frequency (evaluated as the ratio between the number of executed rollbacks and the total number of executed events) and the average rollback length (evaluated as the ratio between the total number of rolled back events and the number of executed rollbacks), whose combination determines the amount of rollback, and measures related to the average granularity of the rolled back events.

An important aspect associated with the study of WGS relates to the parameter $MASW$ used at the level of system F. Typically $MASW$ depends on the specific simulation application and also on dynamics associated with its execution, hence cannot be determined a priori in general. For this reason, we will treat $MASW$ as the independent parameter of the analysis. Specifically we will study the behavior of WGS while hypothesizing different values for $MASW$ into a given range. The observation of the rollback pattern (namely rollback frequency and average rollback length) for the different values, and the comparison with the rollback pattern produced by LTF will provide hints on the correctness of our hypotheses. This will also provide the basis for a discussion we develop in Section 6.5 on how to automatically identify reasonable hypotheses for $MASW$ as the simulation model execution proceeds.

We test WGS under the case of both deterministic event granularity, with perfect a priori knowledge on the granularity of each event type, and non-deterministic event granularity, with a priori knowledge on the expected granularity of each event type. Therefore, in our tests, system WoTE has no real task to perform since, in both cases, the (expected) granularity of each event belonging to GC is predetermined and known.

With respect to latter point there is an important observation we would like to bring to the reader's attention. Actually system FD associated with WGS does not really need to know the exact value of the expected granularity of any given event in the set GC. Instead, it only needs to know whether an event e in that set is expected to have smaller granularity as respect to the other events in the set. In practice, FD needs to know only the ordering relation among the expected granularities of different event types. In most real simulations, this relation is likely to be known in advance since it is determined by the semantics associated with each event type. Specifically, in most real world simulations it is likely that some event types have always (or at least almost always) larger granularity as compared to other event types ([5]). Therefore, at the abstraction level considered in this

[5]This is exactly the case of the mobile system simulation application we shall consider in Section 7, where events associated with resource (channel) allocation are typically much costly to simulate as compared to events associated with resource release.

empirical study, assuming a priori knowledge on the expected event granularity is in practice equivalent to assuming that the relation between the granularities of different types of events is known. As said before, this is typically a reasonable assumption. Anyway, for all those simulation models where the relation between granularities is unknown since it depends on both the type of the events and the LP state at the time of the event execution, estimation techniques of the expected granularity (like the one discussed in Section 4.2) based on execution cost samples evaluated at run time represent adequate solutions to be employed at the level of the WoTE system.

Lastly, each reported data point is computed as the average over 10 runs that were all done with different seeds for the random number generation. Also, the data points refer to simulations with at least 2 millions of committed events. The data related to LTF are reported as single marked points on the y axis in the graphics. Actually, these data are practically identical to those observed for WGS when $MASW$ is set to zero.

6.3 Results for Favorable Test Cases

As already mentioned, in this part of the analysis we are interested in the evaluation of WGS for the case of model execution that exhibits non-minimal amount of rollback and that does not suffer from the ARWTS phenomenon (neither due to the situation in case (i) nor due to the situation in case (ii) described in Section 5.1). This should provide favorable conditions to WGS.

To get such conditions we have decided to use a synthetic benchmark deriving as an instance of the PHOLD model presented in [Fujimoto 1990b]. In this model any LP is an almost stateless entity (i.e. the state vector of any LP records only the current LVT value and the seeds for the random number generation) that executes fictitious simulation events. Any event execution causes the production of a new event to be destined to another LP. The destination LP is selected according to a given communication graph among the LPs that defines the specific topology of the model.

We have considered a classical *fully connected* topology, which often appears as a test case in the parallel discrete event simulation literature [Das and Fujimoto 1994; 1997; Quaglia 2001; Ronngren and Ayani 1994b; 1994a; Som and Sargent 1998; Young et al. 1998]. In such a topology, each LP is connected to all the other LPs and any produced event is equally likely to be destined to any other LP.

The model size, i.e. the number of LPs involved in the simulation model, has been fixed at 32 and the model is executed on 8 machines of the cluster environment presented in Section 6.1, with even distribution of the LPs on the machines. Also, we have selected an initial event population of 3 events per LP. The timestamp increment upon forwarding an event is determined according to an exponential distribution with mean $\alpha = 10$ simulation time units. State saving is performed before the execution of any new event.

Initial events in the system are evenly split into three different granularity classes, namely *light*, *intermediate* and *heavy*. When an event is forwarded to another LP, its class does not change, therefore the amount of non-executed events belonging to each class does not change during the simulation model execution. We have decided to consider three granularity classes since it is common in most real world

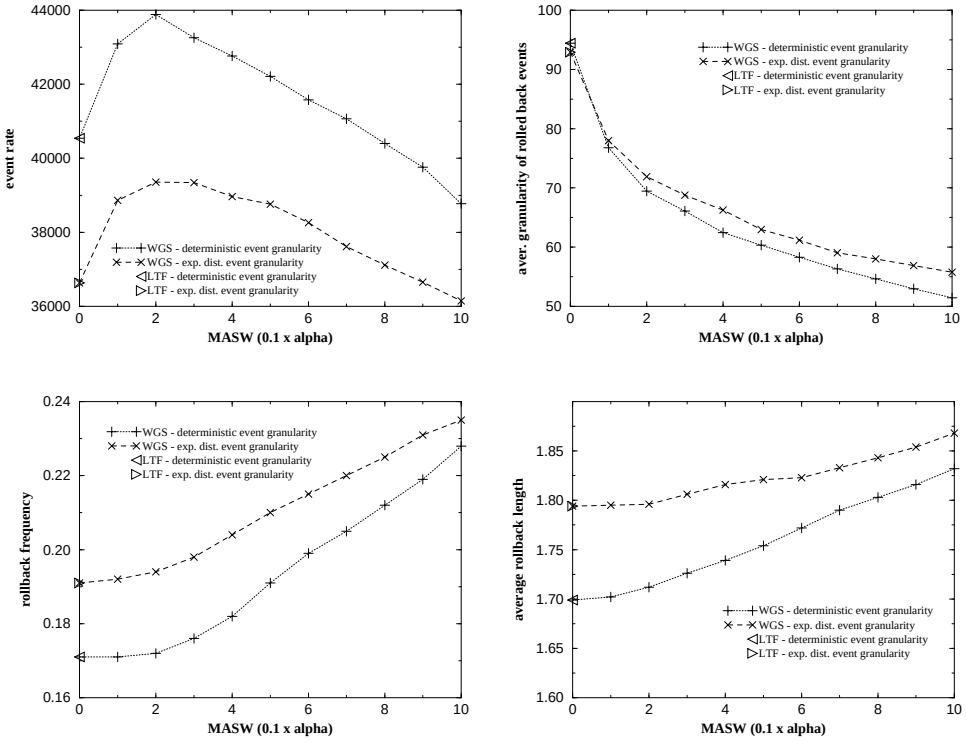

Fig. 4. Results for the fully connected topology.

applications, such as simulations of wireless systems, paging networks and personal communication systems, to have at least two or three distinct event types, each characterized by a distinct granularity. The execution times associated with the three classes are, respectively, 10, 90 and 200 microseconds, which leads to an expected granularity for committed events of 100 microseconds. These granularity values are achieved by busy loops.

As remarked in Section 6.2, we consider both deterministic and non-deterministic event granularity. In latter case, previously mentioned values actually represent expected event execution times and the selected distribution for the execution time of events in each class is exponential ([6]).

Figure 4 shows the obtained results. We observe that for both deterministic and exponentially distributed event granularity the amount of rollback is actually non-minimal (this happens also for the case of LTF scheduling). Specifically, the amount of rollback, expressed as the product between the rollback frequency and average

[6]Exponential distribution of the event granularity is a widely used testing choice. As an example, it has been adopted in the empirical evaluation of the cancelback protocol for memory management in Time Warp synchronization on shared memory architectures presented in [Das and Fujimoto 1997].

rollback length ([7]), is at least in the order of 0.28 for the case of deterministic event granularity and in the order of 0.32 for the case of exponentially distributed event granularity. It increases when values larger than 0.2α are selected for $MASW$. On the other hand, for values of $MASW$ which are less than 0.2α, both rollback frequency and average rollback length exhibited by WGS are practically identical to those exhibited by LTF. This constitutes an empirical evidence of the fact that there exist values for $MASW$ different from zero, for which the $\mathcal{IP}$ property actually holds.

When $MASW = 0.2\alpha$, WGS yields a peak in the event rate providing an acceleration in the simulation model execution, as compared to LTF, which is in the order of 8-10%. Actually, the acceleration derives from the fact that WGS produces a strong reduction in the average granularity of the rolled back events (in the order of 38-33%) with no increase in the amount of rollback.

Note that the acceleration provided by WGS over LTF is slightly larger for the case of exponentially distributed event granularity even though the reduction of the average granularity of the rolled back events is larger for the case of deterministic event granularity. This behavior derives from the fact that, in case of exponential event granularity, we get a larger amount of rollback which increases the final effectiveness of pursuing **Objective 2** in cascade with **Objective 1**.

6.4 Results for Non-Favorable Test Cases

In this part of the analysis we are interested in the evaluation of WGS for the cases of model execution exhibiting minimal amount of rollback or suffering from the ARWTS phenomenon. As already outlined, minimal amount of rollback reduces the effectiveness of pursuing **Objective 2** in cascade with **Objective 1**, while the ARWTS phenomenon reduces the effectiveness of pursuing **Objective 2**, even in case of non-minimal amount of rollback. This should provide conditions that are not favorable to WGS. To get such conditions we have used also in this case instances of the PHOLD model.

The remainder of this section is organized as follows. We first test WGS for the case of a benchmark with non-minimal amount of rollback, but exhibiting the ARWTS phenomenon due to situation in case (i) of Section 5.1, namely relatively longer rollback length. Then we move to a benchmark with non-minimal amount of rollback and short rollback length, which exhibits the ARWTS phenomenon due to situation in case (ii) of Section 5.1, namely an increased number of LPs per machine. Finally we report the results for the case of a benchmark with a reduced amount of rollback.

6.4.1 *ARWTS Phenomenon due to Increased Rollback Length.* To get an execution that exhibits a non-minimal amount of rollback, but that suffers from the ARWTS phenomenon due to an increase in the rollback length, we have used a benchmark similar to the one used in Section 6.3. The only difference is in the connection graph among the LPs, which determines an *hypercube*. Note that also this topology appears in the literature as a test case [Preiss et al. 1994]. Routing is

[7]We recall that the product pr between the rollback frequency and the average rollback length measures the probability for whichever executed event to be eventually rolled back. The quantity $1 - pr$ is a metric referred to as the efficiency of the simulation model execution.

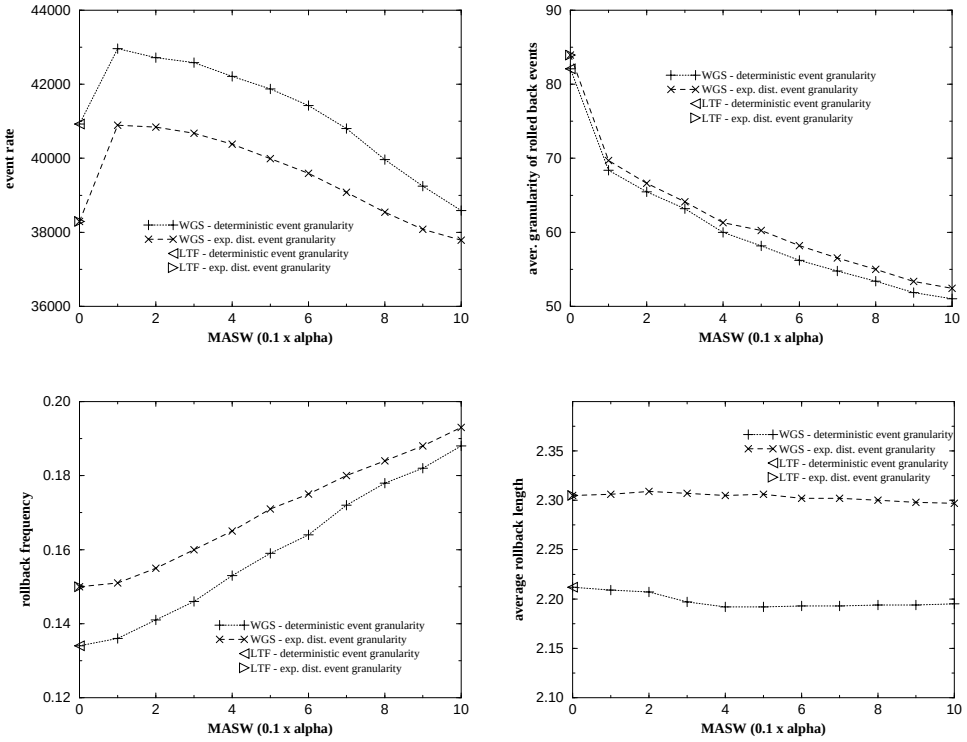

Fig. 5. Results for the hypercube topology.

such that any event at an LP is equally likely to be forwarded to any neighbor of that LP in the hypercube. Also in this case the model is executed on 8 machines, with an obvious mapping of the LPs to machines, and state saving is performed before the execution of any new event.

The results are reported in Figure 5. As compared to those obtained for the fully connected topology in Section 6.3, we get a slight increase in the amount of rollback. Specifically, the amount of rollback is increased to 0.29 for the case of deterministic and to 0.35 for the case of exponentially distributed event granularity.

From such an increase we had expected greater benefits from pursuing **Objective 2** in cascade with **Objective 1**. However, the acceleration in the execution speed provided by WGS over LTF is only in the order of 5-8%, which is slightly lower as compared to the acceleration achieved for the original fully connected model. Such a reduction in the acceleration is due to the fact that changing the topology from fully connected to hypercube modifies the composition of the rollback pattern so that relatively longer rollbacks are obtained. As a consequence the ARWTS phenomenon actually reduces the effectiveness of pursuing **Objective 2**, even in the presence of non-minimal amount of rollback.

To clearly bring to the reader's attention the effects of the ARWTS phenomenon we note that at the point in which the execution speed is maximized (i.e. $MASW =$

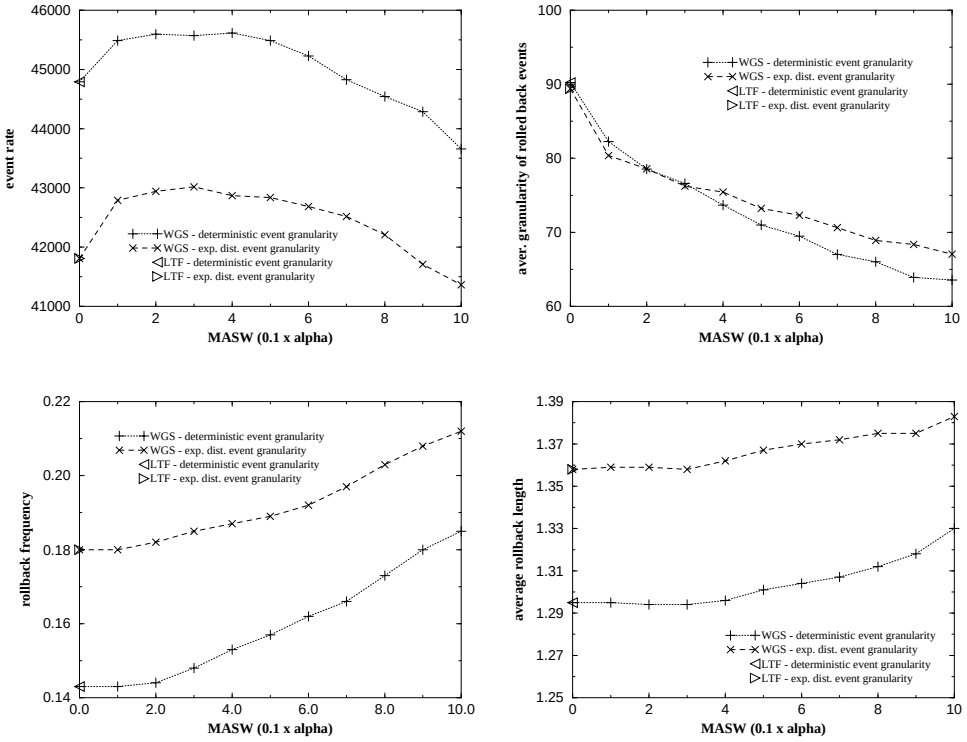

Fig. 6. Results for the fully connected topology with increased model size.

0.1α), the average granularity of the rolled back events is reduced of about 20-22%. Such a reduction is lower as compared to the one obtained at the point in which the performance is maximized for the case of the fully connected topology (i.e. $MASW = 0.2\alpha$) - see Figure 4. This is an indicator that longer rollback length (i.e. manifestation of the ARWTS phenomenon) makes WGS missing a kind of control on the waste of time reduction.

6.4.2 *ARWTS Phenomenon due to an Increased Amount of LPs per Machine.* To get an execution that suffers from the ARWTS phenomenon due to an increase in the amount of LPs per machine, we have simply used the same benchmark used in Section 6.3 with an increased amount of LPs. Specifically, the model size has been increased to 80 LPs, evenly distributed among the 8 machines of the cluster. Actually, increasing the model size with no reduction of the event population would lead to a decrease in the amount of rollback due to reduced parallelism in the model execution. This would hide the effects of the ARWTS phenomenon. To avoid this, we have also reduced the amount of events per LP inserted in the system. Specifically, the initial total event population has been fixed at 81 events, evenly distributed among the three event granularity classes previously mentioned. Also in this case, state saving is performed before the execution of any new event.

The results are reported in Figure 6. In this case we get non-minimal amount

of rollback with minimal rollback length (i.e. in the order of 1.30/1.36 events when $MASW$ is equal to zero). However the performance gain provided by WGS is strongly reduced as compared to that achieved for the original fully connected model analyzed in Section 6.3. Also in this case, the reduction is due to the lower effectiveness of WGS in reducing the average granularity of the rolled back events. Specifically, at the point in which the performance is maximized (i.e. $MASW = 0.2\alpha$ for the case of deterministic event granularity and $MASW = 0.3\alpha$ for the case of exponentially distributed event granularity) the reduction in the average granularity of the rolled back events is only in the order of 8-14%. This is due to the ARWTS phenomenon caused by the increased amount of LPs per machine, which also in this case makes WGS missing a kind of control on the waste of time reduction.

6.4.3 *Reduced Amount of Rollback.* To get an execution exhibiting a reduced amount of rollback we have used the same benchmark used in Section 6.3 with a larger event population. Specifically, the event population has been increased at 9 events per LP. Actually, increasing the event population leads to a relevant decrease in the rollback frequency and to a slight increase in the rollback length, which, combined together, ultimately yield a noticeable reduction in the amount of rollback. Also in this case the initial event population has been evenly distributed among the three event granularity classes previously mentioned and the model has been executed on 8 machines, with state saving performed before the execution of any new event.

The results are reported in Figure 7. They show that, at the point in which performance is maximized (i.e. $MASW = 0.05\alpha$ for the case of deterministic event granularity and $MASW = 0.1\alpha$ for the case of exponentially distributed event granularity), the reduction in the average granularity of the rolled back events is similar to the one achieved for the case of the original fully connected model. This is due to the fact that the rollback length is close to that achieved for the original model, therefore WGS does not significantly miss control on the waste of time reduction due to the ARWTS phenomenon. However, the reduced amount of rollback makes less relevant the impact of the reduction in the average granularity of the rolled back events, which leads to an increase in the execution speed of at most 3-4%.

6.5 Determining a Reasonable Hypothesis for $MASW$

By the previous results, we get that the plots related to the rollback frequency and the average rollback length produced by WGS follow a classical shape. Specifically, for relatively small hypothesized values of $MASW$, the values of both the rollback frequency and average rollback length are practically equal to those achieved with LTF, hence those hypothesized values do not invalidate the basic property which the design of the F system underlying WGS relies on, namely the $\mathcal{IP}$ property.

Instead, when the hypothesized value for $MASW$ is increased beyond a given threshold (which assumes a proper value for any specific simulation model execution), at least one between the previous rollback parameters shows an increase, which is an indication that $\mathcal{IP}$ does not hold anymore. At this point the F system results less effective in pursuing **Objective 1**. Also, as soon as the increase in the

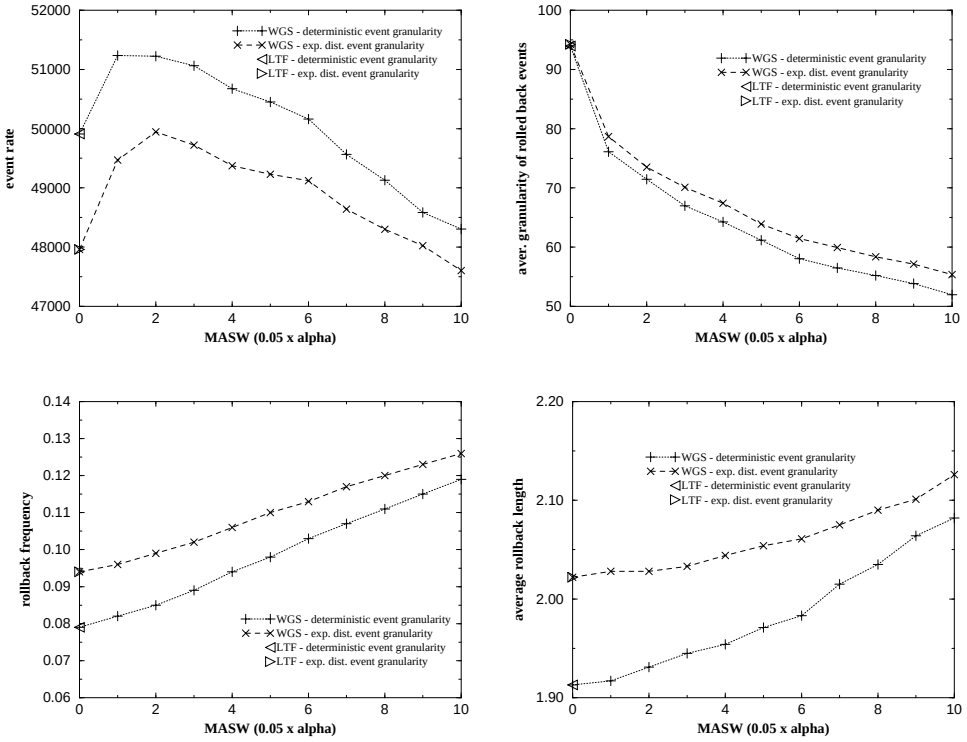

Fig. 7. Results for the fully connected topology with increased event population.

rollback parameters is perceived, the performance provided by WGS degrades.

By the previous arguments we get that the real value to be assigned to $MASW$ should correspond to the threshold value beyond which the $\mathcal{IP}$ property does not hold anymore. On the other hand, such a threshold value is strongly related to dynamics proper of the simulation model execution which, in turn, depends on proper features of the simulation model (e.g. the model size[8]), proper features of the underlying computing system (e.g. the amount of machines), the mapping of LPs to machines and so on. Therefore, the value to be assigned to $MASW$ cannot be determined a priori in general, but should be determined in an application specific manner. As a consequence $MASW$ belongs to the class of the so called *tunable parameters*, whose value should be tuned as the simulation model execution proceeds.

Tunable parameters are typically classified along two dimensions [Srinivasan 1995], namely static vs dynamic and local vs global, and a number of algorithms exist in the literature to cope with the tuning problem of those parameters. The basic principle underlying any of these algorithms is to tune a parameter value on

[8]We will explicitly report data related to the dependency of $MASW$ on the model size in the study related to the mobile system simulation in Section 7, where a wide range of values for the amount of LPs involved in the simulation will be considered.

the basis of monitoring a proper reference metric. If the parameter is a global one, then the metric the tuning relies on is a global one as well.

In the analysis in Section 6.3 and Section 6.4 we have treated $MASW$ as a global parameter ([9]). We maintain this view point in this section while discussing issues related to automatic tuning. Anyway we would like to underline that no theoretical limitation prevents the possibility to treat $MASW$ as a local tunable parameter since its value is related to the assurance of the $\mathcal{IP}$ property, which is actually defined on a local basis.

The algorithm presented in [Srinivasan 1995] in the context of the tuning problem of a global parameter that characterizes the behavior of a synchronization mechanism with controlled optimism, namely Elastic Time [Srinivasan and Reynolds, Jr. 1998], can be straightforwardly used to tune $MASW$ when considered as a global parameter. The algorithm uses the total rate of commitment of events, namely the event rate, as a global metric. In a first phase, so called *scan phase*, the algorithm identifies an adequate initial parameter value that represents the starting point for the tuning process. Then the truly tuning process actually takes place. We note that the tuning process does not lead to a particular value of $MASW$, instead, it keeps $MASW$ in a region about the maximum value of the performance metric. Therefore, tuning performed by this algorithm actually treats $MASW$ as a global dynamic tunable parameter ([10]).

For the benchmarks used in Section 6.3 and in Section 6.4, we have evaluated the ratio between the event rate that can be achieved with automatic tuning of $MASW$ on the basis of the previous algorithm and the best observed value of the event rate while adopting manually selected different hypotheses for $MASW$. The results, reported in Table 6.5, denote that automatic tuning performs in practice at the same way as manual selection of the best suited hypothesis. Therefore, using such an algorithm allows WGS to be employed effectively with no user effort in providing hints on the value of $MASW$.

	Observed Ratio	
Benchmark	Det. Event Gran.	Exp. Dist. Event Gran.
Fully Connected	0.994	0.989
Hypercube	0.984	0.978
Fully Connected with Increased Model Size	0.992	0.981
Fully Connected with Increased Event Population	0.988	0.992

Table I. Manual vs automatic tuning of $MASW$.

7. A CASE STUDY ON A MOBILE SYSTEM APPLICATION

In this section we report results demonstrating the benefits from WGS for Time Warp simulations of a mobile communication system. As a testing environment we have used the same one described in Section 6.1, with the only difference that the

[9] "Global" means that its value is the same for all the entities involved in the execution. Typically entities coincide with LPs, however, in our case an entity coincides with the scheduler module on any machine.

[10] According to the previously mentioned classification, a static parameter is one that, once tuned, will not require re-tuning during the simulation execution.

packet size handled at the level of the MFM communication layer has been set to 64 bytes in order to fit the requirements of this application.

In a mobile system, base stations provide communication services to mobile units. In our simulation model the service area is partitioned into cells, each modeled by a distinct LP. Each cell represents a receiver/transmitter having either some fixed number of channels allocated to it (Fixed-Channel-Assignment, namely FCA) or a number of channels dynamically assigned to it (Dynamic-Channel-Assignment, namely DCA). We consider an FCA model with 100 channels per cell. The model is call-initiated [Carothers et al. 1995] since it only simulates the behavior of a mobile unit during conversation (i.e. the movement of a mobile unit is not tracked unless the unit itself is in conversation). Therefore, the model is organized around two entities, namely cells and calls. Call requests arrive to each cell according to an exponential distribution [Boukerche et al. 1999; Carothers et al. 2000; Carothers et al. 1995] with inter-arrival time $t_{int} = 2$ seconds. All the calls initiated within a given cell are originated by the LP associated with that cell, therefore no external call generator is used.

There are three types of events, namely hand-off (due to mobile unit cell switch), call termination and call arrival. A call termination simply involves the release of the associated channel and statistics update. The granularity of this type of event is about 15 μs. The granularity of the events associated with call arrival depends on whether the channels at the destination cell are all busy or not. If all channels are busy, the incoming call is simply counted as a block. If at least one channel is available, then signal to interference-plus-noise ratio is computed when allocating the channel. The cost of this operation is dependent on the amount of busy channels at the time of the call arrival, therefore the granularity of this event type may vary from about 20 μs up to about 400 μs. When a hand-off occurs between adjacent cells, the hand-off event at the cell left by the mobile simply involves the release of the channel and statistics update, and has granularity of about 15 μs. Instead, the granularity of the hand-off event at the destination cell varies depending on whether all channels in this cell are busy or not. If there is no available channel, then the call is simply cut off (dropped), otherwise, an available channel is assigned to the call and signal to interference-plus-noise ratio is computed when allocating the channel. Therefore, the granularity for this event type has range similar to that associated with incoming call events. Overall, the events can be split into two different classes for what concerns the expected granularity.

As already said, hand-off takes place each time a mobile unit currently involved in conversation moves from one cell to another. In our model there are two distinct classes of mobile units. Both of them are characterized by a residence time within a cell which follows an exponential distribution, with mean 3 minutes (fast movement units) and 30 minutes (slow movement units), respectively. The average holding time for each call associated with both fast and slow movement units is 2 minutes. When a call arrives at a cell, the type (slow or fast) of the mobile unit associated with the incoming call is selected from a uniform distribution, therefore any call is equally likely to be destined to a fast or a slow movement mobile unit.

We have simulated a mobile system in which each cell is hexagonal, therefore all the cells, except bordering cells of the coverage area, have six neighbor cells.

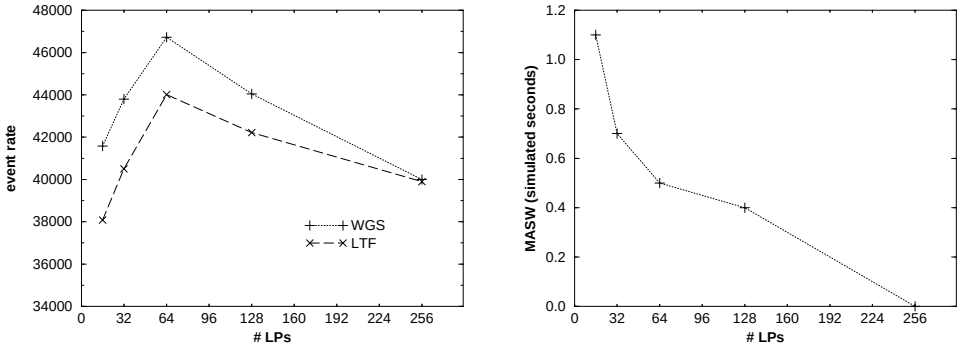

Fig. 8. Results for the mobile system model.

The model size, in terms of number of cells, has been varied between 16 and 256. The corresponding LPs have been evenly distributed among the 8 machines of the cluster. State saving is performed before the execution of any new event.

In this study we present a comparative analysis of the performance, measured in terms of event rate, of WGS and LTF. Therefore, we report the event rate achievable with these two scheduling algorithms while varying the model size. For what concerns WGS, we report, for each model size value, the peak event rate achieved while manually assigning different values to the parameter $MASW$, stepped by 0.1 times t_{int}, and we also report the value of $MASW$ originating the reported peak event rate value. Each reported value results as the average over ten runs, all done with different random seeds. At least 2 millions of committed events have been simulated in each run.

By the results, reported in Figure 8, we observe a tendency that confirms the observations made for the synthetic benchmark based study in Section 6. Specifically, WGS allows improvement in the execution speed (up to 9%) that tends to decrease while the model size increases. Anyway, the gain achievable by WGS still remains in the order of 5% for model size up to 128. Actually, when the model size is further increased, WGS and LTF exhibit in practice the same performance.

A major explanation for the reduced gain of WGS vs the model size is the reduction of the amount of rollback we have observed while increasing the model size itself. As discussed in Section 5, reduced amount of rollback leads to a reduction in the effectiveness of pursuing **Objective 2**. For this specific case, the efficiency of the parallel execution is in the order of 0.75 when the model size is 16 and then increases to a maximum of about 0.97 when the model size is 256, passing through 0.82, 0.88 and 0.93 for intermediate model size values. Besides, we have noted that the rollback length decreases while the model size increases (it starts from about 4 events when the model size is 16 and reaches a minimum of about 2 events when the model size is 256), therefore additional reduction of the effectiveness in pursuing **Objective 2** while the model size increases might be causes by the ARWTS phenomenon due primarily to the increased amount of LPs hosted by any machine.

With respect to the values of $MASW$ originating the peak event rates of WGS, we have that for model size up to 128 this parameter actually assumes values

larger than zero. Instead, when the model size is increased to 256, it assumes the value zero, which means that the $\mathcal{IP}$ property underlying the design of WGS stops holding even for intervals $I(MASW)$ of minimal length ([11]). As a support to this deduction, we have observed that in case of a model size of 256, manually assumed values for $MASW$ larger than zero actually produce a significant increase in the amount of rollback. This is not the case for model size up to 128, for which there exist assumed values for $MASW$ larger than zero that do not impact the amount of rollback of the parallel execution.

8. WGS SCHEDULING VS GLOBAL VIRTUAL TIME

The F system in (3) is based on $\mathcal{IP}$ that expresses the possibility of multiple good candidate events for the execution on a processor on the basis of considerations related only to a local viewpoint. In other words, the decision on whether an event $e \in C$ should be included in GC or not is based only on the local evaluation of the distance between $ts(e)$ and $ts(min(C))$.

However, from a global viewpoint, filtering based only on $\mathcal{IP}$ could make the simulator suffering from the problem we shall discuss below, which we name *delayed GVT advancement* problem.

To explain the problem let us introduce few notations:

$P = \{p_1, \ldots, p_n\}$ denotes the set of the n processors involved in the simulation;

C_i and GC_i denote, respectively, the candidates set C and the good candidates set GC associated with processor p_i.

Let us define Commitment Horizon Processors set (CHP) the following:

$$CHP = \{p_i \mid ts(min(C_i)) = GVT\} \tag{13}$$

In other words, CHP includes each processor p_i, if any, whose set C_i contains an event not yet executed whose timestamp coincides with the current GVT value.

The F system in (3) does not prevent that an event $e \in GC_i$, with $e \neq min(C_i)$ and $ts(e) > ts(min(C_i))$, is selected for the execution on processor p_i. Therefore, in case $p_i \in CHP$, GVT is not advanced until $min(C_i)$ will eventually be selected for the execution on processor p_i.

There are some major implications associated with the delayed GVT advancement problem. Specifically, delaying the advancement of the GVT of the simulation may cause an increase in the distance between the timestamps of executed events and the commitment horizon of the simulation. Also, delaying the advancement of the GVT of the simulation may cause delay on all those operations associated with event commitment, such as memory recovery and displaying of intermediate simulation results.

Some of these implications, e.g. the increased timestamp distance from the commitment horizon and the delay in the execution of memory recovery procedures, may adversely impact the execution speed of the simulation. (Increased timestamp distance from the commitment horizon may lead to an increase in the amount of

[11] Recall that when $MASW$ is set to zero, WGS boils down to LTF.

rollback. Also, delay in the memory recovery procedure may decrease the performance of the underlying virtual memory system.) However, such impact can be controlled by using an algorithm that adjusts $MASW$ on the basis of adequate performance metrics related to the simulation execution speed itself, such as the algorithm discussed in Section 6.5. In other words, the effects of all those implications directly related to the execution speed can be controlled in practice through a correct selection of the parameter $MASW$. On the other hand, the effects of all those implications that are not directly related to the execution speed might not be captured by such an algorithm, and thus might be out of control of the tuning process. As an example of drawback, in the context of interactive distributed simulation, the delayed GVT advancement problem might yield unacceptable latencies in the displaying process of intermediate results to the end-user.

We sketch here two possible approaches to overcome the effects of delayed GVT advancement that are not strictly related to the performance of the simulation model execution. A first approach would be to design an algorithm for tuning $MASW$ relying on metrics that, beyond the performance in the simulation model execution, take into account also the relevance of operations associated with event commitment. On the other hand, a second approach would be to prevent the delayed GVT advancement problem at all, which might yield benefits also to the performance of the model execution itself. This can be done in case the underlying computing system allows efficient calculation and dissemination of globally reduced values [Fujimoto and Hybinette 1997; Srinivasan et al. 1998]. Specifically, if GVT can be computed frequently and efficiently as a globally reduced value, then any processor p_i can be granted at anytime of information to determine whether it belongs to CHP or not. In this case, system F can be modified as follows in order to make any processor $p_i \in CHP$ to select for the execution the event belonging to C_i having the minimum timestamp, namely the event associated with the commitment horizon:

$$\mathrm{F}: \quad \begin{cases} GC_i = \{min(C_i)\} & \text{if } \ p_i \in CHP \\ GC_i = \{e \in C_i \mid ts(e) \in I(MASW)\} & \text{if } \ p_i \notin CHP \end{cases} \tag{14}$$

9. ASSESSMENTS AND CONCLUSION

We have introduced a framework for the processor scheduling problem in Time Warp synchronization, and we have shown how the existing *event-based* scheduling algorithms can be seen as instances of the framework.

This study allowed us to determine a bottom line of the scheduling problem in two main objectives, namely limiting the amount of rollback and the waste of CPU time in the simulation model execution. We have accordingly redefined the success of a processor scheduling algorithm as the capability to effectively pursue both objectives, which is a refinement of the classical definition of a "good" algorithm as the one limiting exclusively the amount of rollback. A rational deployment of the objectives' achievement along the framework provides well-defined semantics and role to each framework block (Filtering (F), Waste of Time Evaluation (WoTE), and Final Decision (FD)). This constitutes a valuable feature for the framework usability.

In order to show that the combination of a low amount of rollback and a low waste of CPU time may actually improve the performance achievable by Time Warp synchronization, starting from the framework we have instantiated an algorithm, namely WGS, explicitly designed to pursue both objectives.

Besides the instantiation of WGS, we have also conducted a theoretical analysis on the framework and we have characterized two circumstances in which any algorithm that results as an instantiation of the framework is expected to be relatively ineffective. Specifically, we have identified, and discussed, a common cause to both these circumstances, which goes under the name of *Aging of Relevant Waste of Time Sources* (ARWTS). Beyond ARWTS, we expected any algorithm deriving from the framework to heavily gain in performance when the real waste of time in the simulation model execution is a non-negligible percentage of the simulation execution time, namely when, in spite of the effectiveness of the processor scheduling algorithm, the amount of rollback in the simulation keeps non-minimal.

We have tested the WGS algorithm against classical synthetic benchmarks in both theorized advantageous and non-advantageous assets. In the analysis we have considered several performance indices: the event rate, which acts as the primary performance index, the average granularity of rolled back events, the rollback frequency and the average rollback length, which allowed us to observe the effectiveness of the joint actions of the framework blocks. All these quantities have been observed while varying the width of a *scheduling window*, that acts as the tunable parameter of the WGS algorithm.

In all the considered configurations the event rate shows a concave shape. The straight meaning of this is as follows: there is an increasing performance gain while increasing the scheduling window (as opposed to a null window where our algorithm behaves exactly as LTF scheduling does) up to a maximum value, and beyond that value the performance irreversibly degrades. The best performance gain (in percentage with respect to LTF) actually appears in case of non-minimal amount of rollback in the model execution, while the performance peaks are lower in the cases of both scarce amount of rollback and ARWTS occurrence. This fully supports the previously mentioned theoretical analysis.

Additionally, we have reported a performance study of WGS for the case of a mobile system simulation application in order to determine the benefits achievable in case of a real world simulation model. In this study, we have varied the simulation model size, in terms of number of LPs, within a quite wide range of values.

We have also identified a relation between WGS scheduling and the advancement of GVT. Specifically, the processor(s) hosting the non-executed event(s) whose timestamp coincides with GVT (if any), due to the algorithm structure, could (for an unpredictable number of times) favor other events in the scheduling decision, thus delaying the advancement of GVT. The impact of such relation on performance and on event commitment related operations (e.g. interaction with external devices, such as displaying of intermediate results in interactive environments) has been discussed. The discussion has also pointed out directions for possible solutions.

This work, besides representing an unifying view of existing *event-based* processor scheduling algorithms for Time Warp synchronization, provides a framework of reference for new algorithms. The deployment of scheduling actions along the

framework helps to evaluate the effectiveness of every single action (i.e. filtering, waste of time evaluation and final decision). The WGS algorithm that we have presented is the first example of an algorithm pursuing both objectives here claimed. Finer filtering actions and/or waste of time evaluation systems taking into account CPU time contributions neglected in the design of WGS may be envisaged to give rise to more effective processor scheduling algorithms.

REFERENCES

Boukerche, A., Das, S. K., Fabbri, A., and Yildz, O. 1999. Exploiting model independence for parallel PCS network simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 166–173.

Carothers, C. D., Bauer, D., and Pearce, S. 2000. ROSS: a high performance modular Time Warp system. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 53–60.

Carothers, C. D., Fujimoto, R. M., and Lin, Y. B. 1995. A case study in simulating PCS networks using Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 87–94.

Das, S. R. and Fujimoto, R. M. 1994. An adaptive memory management protocol for Time Warp parallel simulation. In *Proceedings of Sigmetrics'94*. ACM, New York, 201–210.

Das, S. R. and Fujimoto, R. M. 1997. An empirical evaluation of performance-memory trade-offs in Time Warp. *IEEE Trans. on Parallel and Distributed Systems 8,* 2 (Feb.), 210–224.

Ferscha, A. 1995. Probabilistic adaptive direct optimism control in Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 120–129.

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

Fujimoto, R. M. 1990a. Parallel discrete event simulation. *Communications of the ACM 33,* 10 (Oct.), 30–53.

Fujimoto, R. M. 1990b. Performance of Time Warp under synthetic workloads. In *Proceedings of the Multiconf. on Distributed Simulation*. Society for Computer Simulation, 23–28.

Fujimoto, R. M. and Hybinette, M. 1997. Computing global virtual time in shared-memory multiprocessors. *ACM Trans. on Modeling and Computer Simulation 7,* 4 (Oct.), 425–446.

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

Hamnes, D. O. and Tripathi, A. 1994. Evaluation of a local adaptive protocol for distributed simulation. In *Proc. 1994 International Conference on Parallel Processing*. CRC, 127–134.

Jefferson, D. R. 1985. Virtual time. *ACM Trans. on Programming Languages and System 7,* 3 (July), 404–425.

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

Mascarenhas, E., Knop, F., Pasquini, R., and Rego, V. 1998. Minimum cost adaptive synchronization. *ACM Trans. on Modeling and Computer Simulation 8,* 4 (Oct.), 401–430.

MYRICOM. 1999. Lanai 4. *Draft*.

Pakin, S., Lauria, M., and Chen, A. 1995. High performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet. In *Proc. of Supercomputing'95*. ACM/IEEE Computer Society.

Palaniswamy, A. C. and Wilsey, P. A. 1994. Scheduling Time Warp processes using adaptive control techniques. In *Proc. of 1994 Winter Simulation Conference*. Society for Computer Simulation, 731–738.

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

Preiss, B. R., MacIntyre, D., and Loucks, W. M. 1992. On the tradeoff between time and space in optimistic parallel discrete-event simulation. In *Proc. of the 6th Workshop on Parallel and Distributed Simulation*. Society for Computer Simulation, 33–42.

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

Quaglia, F. 2001. A cost model for selecting checkpoint positions in Time Warp parallel simulation. *IEEE Trans. on Parallel and Distributed Systems 12,* 4 (Feb.), 346–362.

Reynolds, Jr., P. F. 1988. A spectrum of options for parallel simulation. In *Proc. of 1988 Winter Simulation Conference*. Society for Computer Simulation, 325–332.

Ronngren, R. and Ayani, R. 1994a. Adaptive checkpointing in Time Warp. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation*. Society for Computer Simulation, 110–117.

Ronngren, R. and Ayani, R. 1994b. Service oriented scheduling in Time Warp. In *Proc. of 1994 Winter Simulation Conference*. Society for Computer Simulation, 1340–1346.

Som, T. K. and Sargent, R. G. 1998. A probabilistic event scheduling policy for optimistic parallel discrete event simulation. In *Proc. of the 12th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 56–63.

Srinivasan, S. 1995. NPSI adaptive synchronization algorithms for PDES. Ph.D. thesis, Department of Computer Science, University of Virginia.

Srinivasan, S., Lyell, M. J., Reynolds, Jr., P. F., and Wehrwein, J. 1998. Implementation of reductions in support of PDES on a network of workstations. In *Proc. of the 12th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 116–123.

Srinivasan, S. and Reynolds, Jr., P. 1998. Elastic time. *ACM Trans. on Modeling and Computer Simulation 8,* 2 (Apr.), 103–139.

Young, C. H., Abu-Ghazaleh, N. B., and Wilsey, P. A. 1998. OFC: A distributed fossil-collection algorithm for Time Warp. In *Proc. of the 12th International Symposium on Distributed Computing*. LNCS, 408–418.

Received May 2001; revised April 2002; accepted October 2002