

Controlling Bias in Optimistic Simulations with Space Uncertain Events

Valerio Gheri, Giovanni Castellari, Francesco Quaglia
Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma

Abstract

Simulation is becoming an increasingly important technique for what-if analysis in the context of (real-time) decision making applications. Consequently, quick delivery of simulation outputs to end-users (or applications) is a core objective. One approach for high performance simulation consists of exploiting parallel techniques, where the simulation model is partitioned into objects (or Logical Processes), concurrently executing events on different CPUs and/or multiple CPU-Cores. For this type of simulation systems, a further run-time improvement arose from the exploitation of event uncertainty, both in time and space, which has led to more flexible synchronization protocols. Although this approach can provide significant performance gains, one drawback is the risk of less reliable simulation results due to potential bias induced by the mechanisms for resolving the uncertainty, which are sometimes exclusively targeted to run-time effectiveness. In this article we focus on space uncertain simulation events in optimistic parallel simulation and introduce a mechanism that, compared to previous approaches, allows trading-off execution speed vs reliability of simulation results. In other words, our target is the achievement of high performance while controlling, at the same time, the bias introduced by space uncertainty on the simulation output.

1 Introduction

Discrete Event Simulation (DES) is a well known technique to study and predict the behavior of complex systems. Recently, this technique has acquired the role of core component in time critical decision making processes. This has renewed the need for solutions allowing quick delivery of simulation results to end-users or applications.

A traditional way to achieve high performance simulations is the employment of parallelization techniques (hence Parallel DES - PDES) aimed at exploiting parallel (or distributed) computing systems to accelerate the execution of the simulation model [4]. They are based on the partitioning of the model into Logical Processes (LPs) that can execute events in parallel on different CPUs and/or different CPU-Cores. For these techniques, the main obstacle to speedup and scalability is the overhead due to the synchronization

mechanism among the LPs, which is required to ensure causally consistent execution of simulation events.

Relatively recent studies have proposed a kind of cross-layer approach for further enhancing the performance of parallel simulation systems. This is based on exploiting under-specification of either temporal or spatial properties of simulation events at the application level, in order to allow the underlying simulation platform to finalize the occurrence time/point for those events in a way to optimize performance metrics. Under-specification of temporal properties gave rise to temporal uncertainty, which has been originally exploited in the context of federated simulation systems [5]. Basically, a time uncertain event can potentially occur at any instant within a time interval ΔT . Time intervals increase the events concurrency degree thus allowing parallel simulation systems based on conservative synchronization schemes to overcome problems related to poor lookahead [5, 7]. Temporal uncertainty has been also exploited in optimistic parallel simulations [2], where the increased degree of concurrency among simulation events translates into a reduction in the number of rollbacks and, again, into the consequent reduction of the wall-clock time for the simulation model execution.

More recently, under-specification of spatial properties in the DES model gave rise to spatial uncertainty [8]. This type of uncertainty expresses the possibility for a simulation event to occur within a spatial subregion ΔS of the whole simulated system space, rather than in a precise point. In PDES contexts, the advantage from having a space uncertain event is that all the LPs modeling a portion of the region ΔS might take care of the execution of that event (i.e. they are all candidates for event execution). Therefore, the decision on which LP should really execute that event can be taken run-time in a way to reduce synchronization costs. The work in [8] has also shown how to achieve the reduction of such costs in case of optimistic simulation systems. Specifically, a synchronization algorithm called Rollback-If-No-Additional-Chance (RINAC) has been proposed, which selects the actual LP for event execution, via ad-hoc event forwarding schemes among the candidate LPs, in the attempt to reduce the amount of rollback in the parallel execution.

In this work we still cope with space uncertain events, and present a bias control mechanism where forwarding decisions among the candidate LPs rely on statistical data related to where, in the simulated space, already routed uncertain events have been located. These data express how far the simulation results are expected to be from those that would have been obtained by applying pure randomization for the selection of the destination LPs. In other words, we are taking pure randomization as the reference point for unbiased model executions, and we try to provide performance effective event de-routing mechanisms that do not push simulation results far from those that would characterize the reference (unbiased) scenario. We note that our approach is not application specific, thus representing a general solution for the treatment of space uncertain events in DES systems.

The remainder of this paper is structured as follows. In Section 2 we provide some details on RINAC and the associated event forwarding scheme. In Section 3 the bias control mechanism for the treatment of space uncertain events is described. Integration of this mechanism within an operating optimistic simulation platform is provided in Section 4. Experimental data are reported in Section 5.

2 RINAC: Description and Comments

As sketched in the Introduction, RINAC is a routing scheme for space uncertain events in optimistic simulation systems. It aims at reducing the synchronization cost via the reduction of the amount of rollback in the parallel execution. In the original model [8], spatial uncertainty needs to ultimately resolve the location of space uncertain events via the instantiation of a so called *location()* function. In RINAC the instantiation is based on a forwarding mechanism according to which an LP that receives a notification message for a space uncertain event e accepts the event (i.e. incorporates the event in its event queue) in case its simulation clock is not greater than the event timestamp. Otherwise it rejects the event, and the notification message is forwarded to another LP among the possible candidates. This is done in the hope that the next destination LP is able to maintain a so called *acceptability* property in the model execution [8], based on timestamp ordered execution of events at each LP, without the need for rollback procedures.

To avoid cycles of forwarding, that might lead to a situation in which no LP eventually accepts the event (thus the event itself is not eventually executed), forwarding can be done only among the set of LPs not yet visited by the notification message. If all the possible destination LPs have been already visited, the LP lastly visited in the forwarding scheme cannot reject the event, even in case the event timestamp is lower than the current simulation clock of that LP. Therefore, we need to execute rollback on that LP to recover acceptability in the model execution since there is no more chance to visit another LP in the forwarding scheme. Hence the name RINAC (Rollback-If-

No-Additional-Chance). However, the rollback cost is paid only in the adverse situation where no LP along the forwarding chain has a logical clock value not greater than the timestamp of the space uncertain event.

Actually, the complete lack of a whatever control on uncertain events actual receivers (in fact the receivers are selected only on the basis of synchronization requirements and related overhead), places a compromise between performance and statistical validity of the simulation results [6]. Hence, these results could be sensibly biased due to performance targets.

3 The Bias Control Mechanism

3.1 Non-biased Simulations

Spatial uncertainty means that no specification at all is provided for what concerns the exact location of some (or all) the events. Instead, only a plausible region is specified, within which the event will occur. This means lack of specification of the real distribution function, say Φ , for the location of those events.

On the other hand, the mechanism adopted by the simulation platform to resolve the uncertainty gives rise to an actual event distribution, say $A-\Phi$, determined by actual event routing decisions among the candidate LPs taken on the basis of whichever objective. One relevant objective is the attempt to minimize the latency for the model execution.

Given this premise, in principles we can say that non-biased simulations are those for which routing decisions for uncertain events provide an actual distribution such that $A-\Phi=\Phi$. However, if all the uncertain events are located according to such an actual distribution, there is no possibility to exploit event uncertainty for specific purposes since event routing and final location rules need to match a pre-determined statistical criterion. Reversing the reasoning, by using techniques for convenient routing and localization of uncertain events, there should be no possibility to fully accommodate the Φ distribution.

What we introduce in the next section is a method for locating space uncertain events, which provides an actual distribution $A-\Phi$ close to Φ , namely with distance bounded by a constant value (which will be referred to as *EMAX*). We do this for the particular case where no specification at all about the position of uncertain events is interpreted as having pure randomization for the selection of occurrence points of uncertain events within the plausible regions. In other words, we consider the case in which Φ is a uniform distribution function.

3.2 The Event Routing Algorithm

As said, we focus on a uniform Φ distribution. Also, for simplicity of presentation we assume that all the candidate LPs modeling the plausible region for the occurrence of an uncertain event have the same weight, in terms of the size of the modeled portion of the plausible region. If this is

not the case, then the event routing mechanism we present in this section would require the employment of weighting (corrective) factors expressing the different relevance of the candidate LPs within the plausible region.

In our proposal the candidates' set represents the fundamental information unit. In fact we handle event routing decisions by exploiting per-set information. As an example, if a given LP, say LP_A , has accepted 20 uncertain events all associated with the same candidates' set $\{LP_A, LP_B, LP_C\}$ and 10 uncertain events all associated with the candidates' set $\{LP_A, LP_B\}$, our mechanism explicitly distinguishes among the two groups of accepted events for statistical inference. We consider candidates' sets as non-ordered collections of LP identifiers. Hence two candidates' sets are identical if they are composed by the same elements, independently of their occurrence order. Examples of identical sets are $\{LP_A, LP_B, LP_C\}$ and $\{LP_B, LP_C, LP_A\}$.

When a space uncertain event is produced, the corresponding candidates' set is defined, and the eventually selected final destination can be seen as identified according to the following different possibilities. It can either coincide with the LP in the candidates' set that would have been selected by applying a uniform distribution, or it can coincide with an LP in that set selected by a different event routing mechanism. According to this consideration, we can separate space uncertain events into two disjoint subsets, which we call *U-Compliant* (Uniform Compliant) and *U-Uncompliant* (Uniform Uncompliant), respectively. All the events in the U-Compliant subset have their locations uniformly distributed by definition. Hence the objective of our event routing algorithm is to locate all the other events, namely those within the U-Uncompliant set, according to a distribution close to the uniform one. This distribution should anyway be oriented to optimize some performance metric in the model execution.

3.2.1 Data Structures

Each simulation kernel, possibly hosting multiple LPs, keeps a table for each candidates' set. This table has a row for each LP in the set, which keeps track of information about the density of U-Uncompliant uncertain events (from now on just named density). This is defined as the number of U-Uncompliant uncertain events (associated with that candidates' set) which have been accepted by the LP during a simulated time interval. Different simulation time intervals are associated with different columns within the table.

Each time an U-Uncompliant uncertain event is accepted for processing at a given LP (i.e. the event is enqueued into the LP event list), the hosting simulation kernel updates the corresponding entry within the density table. At the same time, the different simulation kernel instances exchange the current values of these tables on a periodic basis. There-

fore, for all the locally hosted LPs, the density table contains exact information about the amount of accepted U-Uncompliant uncertain events within each simulation time interval. On the other hand, the rows associated with remotely hosted LPs record approximate density values for U-Uncompliant uncertain events that have been already accepted by these LPs in the different simulation time intervals.

3.2.2 Base Event Routing

When an uncertain event E is produced and is delivered to the simulation kernel for routing towards the destination, one LP is randomly selected within the candidates' set, say LP_{random} , and the event is routed towards the kernel hosting this LP, which we refer to as K_{random} . Upon the receipt of E at K_{random} , if the local clock of LP_{random} is not greater than the event timestamp, E is accepted for processing at LP_{random} . This also means that E 's destination complies with the uniform distribution, hence E ultimately falls within the U-Compliant set of events. In this case, the event density table at K_{random} does not require to be updated, since it only keeps information about U-Uncompliant events.

We note that, if U-Compliance is verified for all the space uncertain events, then the routing algorithm provides an event distribution $A-\Phi$ identical to Φ , the latter being the uniform distribution. Hence no bias at all is introduced according to the definition of unbiased simulations provided in Section 3.1.

On the other hand, if the local clock of LP_{random} is greater than the timestamp of E , like in the original RINAC proposal, an event de-routing strategy can be adopted in the attempt to avoid the execution of rollback procedures (for this scenario we also say that the event E is currently rejected by LP_{random}). In this case, the final destination for E will not ultimately comply with the uniform distribution since we are deviating from the original random destination choice. Hence the event E will ultimately fall in the U-Uncompliant set. However, differently from RINAC, the de-routing strategy has the objective to provide an actual distribution of U-Uncompliant events which is close to the uniform distribution Φ .

3.2.3 De-routing Strategy

Upon rejection of the event E originally destined to LP_{random} , the hosting kernel K_{random} identifies the simulation time interval the event timestamp falls inside. Then it can identify, via the density table, two different LPs, say LP_{max} and LP_{min} , belonging to the candidates' set of E . (Recall that E is uncertain, hence its candidates' set has cardinality greater than one.) LP_{max} is the candidate receiver having, according to the information currently available at

K_{random} , the highest density of U-Uncompliant uncertain events (destined to that same candidates' set) within that simulation time interval, say d_{max} . Similarly, LP_{min} is the candidate receiver with the lowest density in that same interval, say d_{min} . We note that LP_{random} might coincide with either LP_{max} or LP_{min} . In any case we denote with d_{random} the current U-Uncompliant event density for LP_{random} within the simulation time interval of interest.

The de-routing strategy for the event E is such that the maximal excursion across the density values for that simulation time interval, say $(d_{max} - d_{min})$, must not exceed a pre-specified threshold $EMAX$. A way to achieve this is to de-route the event E towards LP_{min} . In this case, the distance among the densities of different LPs would shrink as d_{min} would grow by one unit. However, in case $LP_{random} = LP_{min}$, a different de-routing decision needs to be taken (since LP_{random} is currently rejecting the space uncertain event E). In such a case, the target LP selected for event de-routing is some LP_j in the candidates' set, which is identified on the basis of increasing values of U-Uncompliant event densities. In particular, LP_j will be associated with the minimum density value d_j which is greater than d_{min} .

A special case occurs when $d_j = d_{max}$, namely when $LP_j = LP_{max}$. In such a case the event E is de-routed towards LP_{max} only in case $(d_{max} - d_{min}) < EMAX$. Otherwise, LP_{random} is forced to finally accept the event E .

In the general case where the number of candidate receivers has cardinality greater than 2, LP_{random} might not coincide with LP_{min} , and LP_j might not coincide with LP_{max} . In such a case, LP_j represents some LP with an intermediate density of U-Uncompliant events within that simulation time interval, and upon the receipt of the event E at the corresponding simulation kernel, a similar density based de-routing strategy is applied in case the timestamp of the event E is lower than the current simulation clock at LP_j (again in the attempt to avoid rollback). Obviously, to avoid cycles within the de-routing algorithm, we do not allow de-routing towards an already visited candidate destination. Hence, the event is eventually processed.

Another fundamental aspect of our de-routing strategy is the aim at diffusing updated information about current density values in order to improve the knowledge on the real system state. This is done at low cost via the exploitation of de-routing messages. In particular, each time an event E is de-routed by LP_j towards a different candidate destination, a reference density value is piggy-backed on the de-routing message. It can either be the current density value d_j associated with LP_j which is sending out the message, or the original value received in piggy-back, depending on which of the two values is the minimum. In other words, when a message notifying an uncertain event E traverses

a sequence of LPs due to de-routing, the minimum density value across all the traversed LPs is spread in order to update kernel level density tables. This is relevant since the receipt of an updated value for the minimum density across the set of candidate receivers can allow the simulation kernel to gain information about the fact that d_{min} has been increased across density tables, thus opening new de-routing possibilities on the basis of the updated value of $(d_{max} - d_{min})$. Specifically, a new de-routing possibility is gained towards LP_{max} in case the updated value for d_{min} tells that currently $(d_{max} - d_{min}) < EMAX$.

Obviously, when the notification message is sent out by the LP that has scheduled the uncertain event, no chain of potential receivers has been traversed yet. Hence, in order to really capture the minimum density value along the chain of touched receivers, the special value $MAXINT$ is initially piggy-backed on the notification message.

In Figure 1 we show the complete pseudo-code for the de-routing algorithm described above. This includes the **SendOut** part, for initially inserting the uncertain event within the system, and the **AcceptOrForward** part, for either accepting or de-routing the uncertain event. As explained, the $MAXINT$ value is initially piggy-backed on the event notification message as the reference density (see line 4), which is then updated according to the aforementioned rule to keep track of the actual minimum observed along the chain of traversed LPs (see line 13). Also, in the **AcceptOrForward** part, the need for de-routing is evaluated on the basis of whether the timestamp of the uncertain event is lower than the local clock of the current recipient LP (see line 7). In this case, the aim of de-routing is exactly the attempt to avoid rollback. On the other hand, the event is accepted (even in case of rollback) either if there are no possible additional destinations (see line 7), or if the residual destination with minimum density does not comply with bounded bias criteria (see line 14 for the compliance test). This occurs when the residual destination with minimum density has anyway processed, in that same simulation time interval, at least $EMAX$ events more than the LP associated with the minimum density across the already traversed potential destinations.

Another important aspect is the treatment of d_{max} , and more in general of the whole set of values kept by density tables. Specifically, a kind of (lazy) update of those values is anyway required in order to provide the distributed simulation kernel instances with the possibility to operate on data really representative of the current system state. Our lazy policy for the notification of local density tables to remote kernel instances is such that a notification message is sent out in case the local density table has been updated in a way that can really affect de-routing decisions. This occurs when the difference $(d_{max} - d_{min})$ locally recorded by the density table approaches $EMAX$. In fact, in such

SendOut:

1. Upon the receipt of $[E, CandidateReceiversSet]$ from the application
2. randomly extract $LP_{random} \in CandidateReceiversSet$;
3. $AlreadyVisited = \emptyset$;
4. $ReferenceDensity = MAXINT$;
5. send $EventNotification[E, LP_{random}, CandidateReceiversSet, AlreadyVisited, ReferenceDensity]$ to Kernel instance hosting LP_{random} ;

AcceptOrForward:

6. Upon receipt of $EventNotification[E, LP_i, CandidateReceiversSet, AlreadyVisited, ReferenceDensity]$ from whichever Kernel instance
 7. **if** $(E.timestamp \geq LP_i.clock)$ OR $(CandidateReceiversSet = AlreadyVisited + LP_i)$ /no straggler or no possibility to de-route
 8. enqueue event E for processing at LP_i ;
 9. **else**
 10. retrieve density table DT associated with $CandidateReceiversSet$
 11. $AlreadyVisited \leftarrow AlreadyVisited + LP_i$; /updating the already visited set to include LP_i
 12. find $LP_j \in (CandidateReceiversSet - AlreadyVisited)$ with minimum density d_j within DT ; /identification of a de-routing destination
 13. $ReferenceDensity = \min(ReferenceDensity, d_j)$; /new minimum across all the visited LPs, including the current receiver LP_i
 14. **if** $(d_j - ReferenceDensity < EMAX)$ /de-routing is possible with bounded bias towards the identified destination
 15. send $EventNotification[E, LP_j, CandidateReceiversSet, AlreadyVisited, ReferenceDensity]$ to Kernel instance hosting LP_j ;
 16. **else** enqueue event E for processing at LP_i ; /no de-routing possible without bias, event is finally accepted even if it is a straggler
-

Figure 1. De-routing Algorithm Pseudo-code.

a case there has been some local LP, namely LP_{max} that (to the local kernel knowledge) has accepted substantially more uncertain events compared to some other LP in the same potential receivers set. Hence, remote simulation kernel instances must be notified about the large density value of U-Uncompliant uncertain events occurred at LP_{max} , so to avoid de-routing of additional uncertain events towards LP_{max} , in that same simulation time interval. On the other hand, if a remote kernel, say K_j , keeps a more recent value for d_{min} , it can anyway decide to forward additional uncertain events to LP_{max} , since the kernel hosting this LP, say K_i , shows to have a pessimistic (obsolete) view of the current value of the difference $(d_{max} - d_{min})$. Actually, a timeout mechanism for the notification of local density tables to remote kernel instances can be used to help reducing communication overhead by preventing multiple notifications within a same timeout period.

We note that, under the ideal setting of instantaneous communication, exceeding $EMAX$ with the density difference across potential receivers due to de-routing decisions is prevented. This is because, as soon as such a threshold difference is suspected on the basis of the currently known value of d_{min} , then LP_{max} is immediately excluded by the de-routing strategy as receiver of the uncertain event.

Relaxing the assumption of instantaneous communication, we can still expect the bias, namely the difference between $A-\Phi$ and Φ for U-Uncompliant events, to be bounded over the long period because the de-routing strategy tends to keep all the LPs within a certain distance in terms of processed uncertain events. On the other hand, for small simulation time intervals we may get that, while an update message is in transit, a kernel, say K_i , could forward an uncertain event towards LP_{max} , with the current value of d_{max} such that $(d_{max} - d_{min}) > EMAX$ since K_i is not yet aware of such a system situation, which will be eventually

notified by the in transit message. Such a decision would otherwise not be taken in case of instantaneous communication. However it is expected to have significant impact on the bias only in case of a burst of de-routing occurrences towards that same LP.

4 Integration within an Operating Simulation Environment

In this section we concisely describe some aspects related to an implementation of the de-routing strategy of space uncertain events within an operating optimistic simulation environment, namely the ROME OpTimistic Simulator (ROOT-Sim). This is a general purpose simulation software based on a simulation kernel layer that ultimately relies on MPI for data exchange across different kernel instances. This software transparently supports all the mechanisms associated with parallelization (e.g. mapping of the LPs on different kernel instances) and optimistic processing. It also offers advanced facilities for the evaluation of global predicates (e.g. global termination conditions) on the committed portion of the simulation [3].

In this platform, the interaction between the application software and the simulation kernel mainly occurs via:

- A callback service to be offered by the application level software, namely `ProcessEvent()`, acting as the event handler.
- Another callback service offered by the application level software, namely `CheckTermination()`, used for global predicates evaluation.
- A service offered by the underlying simulation kernel called `ScheduleNewEvent()`.

The `ProcessEvent()` callback has a set of parameters identifying the event to be processed (in the form of

an application defined data structure), the global identifier of the scheduled LP (expressed as a numerical code used as an LP indexing information) and the base state address associated with the LP for which the event is occurring. It also has an additional parameter acting as a flag, which indicates whether we are in the presence of the first call to `ProcessEvent()` for a given LP. This flag can be exploited for initialization purposes, which might entail initialization of the main data structure representing the LP state. `ProcessEvent()` can be programmed in standard C technology, with the possibility to use `malloc` compliant allocation/deallocation thanks to the integration of the simulation kernel with a dynamic memory logger and restorer library specifically designed for optimistic simulation kernels exhibiting this type of software layering [9]. The `ScheduleNewEvent()` service offered by the platform can be invoked during event processing for injecting new events within the system. This service only needs to receive in input the new event content and a numerical code identifying the destination LP within the whole set of active LPs (whose size is established via an application level defined macro). Finally, the `CheckTermination()` callback receives a committed snapshot from the kernel layer (this occurs when a new Global Virtual Time - GVT - is evaluated), in order to verify a termination predicate (which must be of a stable type).

To support the uncertain event de-routing mechanism, the simulation kernel API exposed to the application level software has been augmented with the service `ScheduleNewSpaceUncertainEvent()`, having similar parameters to the aforementioned event scheduling service, but with the possibility to identify a set of potential receivers (instead of a single LP). On the other hand, the input/output event queue manager has been augmented with modules implementing the de-routing strategy.

In order to achieve a flexible implementation of de-routing services within the simulation kernel, we have decided not to impose predefined sets of potential receivers. Instead, the kernel layer dynamically detects new sets of potential receivers to be managed (on the basis of application level invocations to `ScheduleNewSpaceUncertainEvent()`, and related parameter values), and dynamically allocates event density tables maintaining the information required by the de-routing strategy.

Beyond flexibility, another important point is lightness while managing data structures. In particular, fast access to the density table of interest is a critical aspect. To this end, retrieval of the density table memory address is based on a hash table whose keys are sets of LP identifiers. The associated implementation has been based on the open source TCL library [1].

In order to further optimize the management of the den-

sity tables, by keeping as low as possible the amount of allocation and de-allocation operations, we have decided to manage the table entries associated with the different simulation time intervals, over which density values are kept, as a circular buffer. In particular, when the entry corresponding to a given interval of simulation time gets obsolete (due to GVT advancement beyond the upper extreme of that interval), it is reset and gets associated with a (far in the future) virtual time interval according to the circular policy. The number of entries initially allocated depends on the value of a compile time macro. However, in the adverse scenario where this number does not suffice for keeping track of density values across the whole set of uncommitted simulation time intervals, then a simple re-allocation operation of the density table is executed at run-time. In our implementation, re-allocation has been implemented by doubling the size of the table, thus reducing the likelihood of additional re-allocations within a short time interval.

A final aspect to be considered when dealing with scenarios where events can be de-routed is the treatment of anti-messages, namely negative messages used to cancel both traditional and space uncertain events in case the LP that scheduled those events is rolled back. For space uncertain events, the treatment of anti-messages must keep into account the fact that the original message carrying the event to be cancelled, might have been accepted by some LP after a sequence of potential receivers has been explored during de-routing. In this case, given that de-routing decisions are taken by the intermediate simulation kernel instances managing those LPs, then the kernel level software must keep track of whether the uncertain event has been accepted by some local LP, or has been de-routed. In the latter case, the kernel must also keep track of the identity of the destination kernel instance for the de-routing, since, in case of the arrival of the corresponding anti-message, it must be de-routed along the same path. To achieve high efficiency while managing information about de-routed events, in order to, e.g., fast determine the original path over which the message has been sent, we have used again a hashing mechanism. It relies on a hash table whose keys are message/anti-message identifiers, and whose entries simply record the destination kernel for the original message.

Actually, our software platform has been designed in order to be able to handle non-FIFO message delivery (independently of the fact that MPI actually provides FIFO message delivery along a same channel, i.e., along a sequence of messages with the same tag). Hence, it is possible that an anti-message (either for a traditional or for an uncertain event) gets delivered before the corresponding positive message. In this case the hash table will simply notify that no recipient is associated with the original message. Hence the anti-message is inserted by the kernel into a queue of pending anti-messages, which will be used (as in classical

scenarios not entailing space uncertainty) for internal message annihilation upon the arrival of the original positive message.

5 Experimental Results

In this section we present some experimental data for the analysis of both the achievable performance improvements when exploiting space uncertainty, and the effects of uncertainty on the statistics associated with the simulation model (namely the bias). Our test-bed is a rescue/exploration-like applicative scenario in which the application level simulation code models a square bi-dimensional region, where some phenomenon (e.g. a fire event) is evolving over simulation time, and robot agents move across the region to gain information on the real state of the phenomenon. The region is partitioned into square subregions, each one modeled by a different LP. The agents move over the area of interest according to the random-way point mobility model. Each way-point represents a decision point for planning a new straight path towards the successive way-point, which might depend on specific terrain (or other) conditions characterizing the area of interest. At the same time, the way-point models the destination for the next observation of the phenomenon by the agent. The size of the area of interest is $36 \times 36 \text{ Km}^2$, with each subregion being a square of $6 \times 6 \text{ Km}^2$. The number of agents exploring the area of interest is 4.

Space uncertainty in this model deals with the definition of the position of the way-points characterizing agents mobility. Specifically, the mobility model does not define exact way-points. Instead, it defines a square region of size $1 \times 1 \text{ Km}^2$ the way-point can be located within. This can express unprecise knowledge of whether the planned straight path between two subsequent observation points can be perfectly followed. In particular, some un-know (and hence non-modeled) environmental condition might slightly deviate the original agent plan to reach the target observation point.

Concerning application level statistics, the metric of interest is the amount of simulation time required for a given coverage (say 95% coverage) while observing the phenomenon over the whole area of interest. Concerning the agent ability to observe the phenomenon, we set the size of the observable region at each way-point at $1 \times 1 \text{ Km}^2$.

In each subregion over the area of interest (namely at each LP), the observed phenomenon has been set to evolve over simulation time according to an exponential distribution, with mean value 60 minutes. Concerning agent mobility, the time interval for reaching the next way-point has also been set as exponentially distributed, but with average time 10 minutes.

All the runs have been carried-out on an SMP machine equipped with 4 Xeon CPUs (2.0 GHz) and 4 GB of RAM memory, running LINUX (kernel 2.6). Four instances of

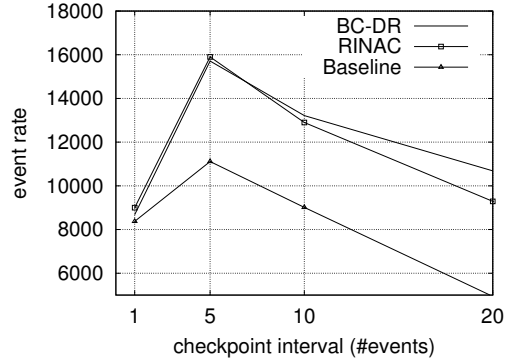


Figure 2. Event Rate vs the Checkpoint Interval.

the ROOT-Sim kernel have been activated on this machine, each hosting 9 of the 36 total LPs. Also the state of each LP has been fixed at 50KB (with state saving cost of about 120 microseconds on the used architecture), while the CPU cost for simulating the events associated with the evolution of the phenomenon over the area of interest has been set at about 100 microseconds.

To assess the performance gain achievable when space uncertainty is exploited, we have measured the event rate, i.e. the number of committed simulation events per wall-clock time unit. The event rate achieved via our de-routing strategy for handling space uncertainty (which we label as Bias-Controlled De-Routing - BC-DR), has been compared against the event rate achieved via the following two different configurations:

- (i) A baseline configuration where space uncertainty is not exploited, therefore the way-point arrival event is always routed towards the LP hosting the center of the square uncertainty region. This baseline represents the case where pure randomization is applied for the determination of way-point locations.
- (ii) The original RINAC de-routing algorithm, where there is no explicit attempt to control bias, but the target is pure performance.

The event rate curves are drawn in Figure 2, where the x-axis expresses the variation of the checkpoint interval. This is the typical parameter determining the actual trade-off between the state saving overhead and the recovery latency. By the curve, we observe that both RINAC and BC-DR provide definitely better event rate compared to the baseline configuration, especially at the point where the performance is maximized vs the checkpoint interval. In other words, they show a definitely better ability to commit simulation events in each unit of elapsed computation time.

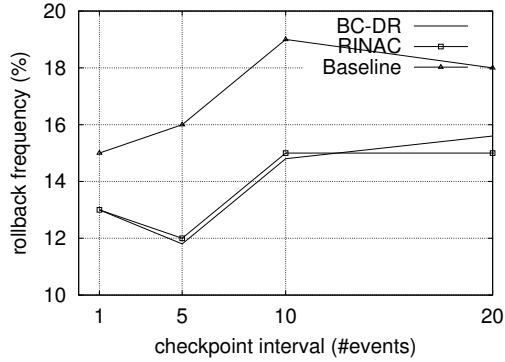


Figure 3. Rollback Frequency vs the Checkpoint Interval.

The main motivation for this result is in the significant reduction of the rollback frequency (¹) achieved by both RINAC and BC-DR. Specifically, as reported in Figure 3, the rollback frequency is reduced by about 30% at the point where the performance (i.e. the event rate) is maximized vs the checkpoint interval. This is an expected behavior, since the objective of both RINAC and BC-DR is the reduction of the amount of rollback thanks to uncertain events' de-routing in case the current recipient would require rollback handling. The reduction of the rollback frequency is also the reason why, looking back at the event rate curve, both RINAC and BC-DR exhibit relatively adequate performance even for very large, non-optimal values of the checkpoint interval. Being the rollback frequency lower than for the baseline case, there is less risk of performance decrease in case an excessively large checkpoint interval is selected. This is because the rollback cost, in terms of coasting forward latency for the reconstruction of an un-checkpointed state value, has low relative impact on the overhead, due to less frequent execution of rollback procedures per wall-clock time unit.

Let us now discuss the effects of the exploitation of spatial uncertainty by RINAC and BC-DR on the statistics at the level of the simulation model, and therefore on the total amount of wall-clock time required for achieving the same precision in the modeled statistical phenomenon. We note that higher event rate only means higher ability to commit simulation work in a wall-clock time unit. However, the final performance indexes must also be based on the observation of how useful is the committed work for converging towards the application level statistics we are interested in. In un-biased simulations, such a work should exhibit better usefulness, hence the baseline scheme is expected to converge towards the final statistic, i.e. the 95%

¹Evaluated as the ration between the number of rollback occurrences in the parallel execution and the number of processed events, committed plus rolled-back.

Table 1. Simulation and Wall-clock Times.

Test-Case	Simulation Time (minutes) for 95% Coverage	Best Case Wall-clock Time (msec)
Baseline	44700	6939
RINAC	57208	6325
BC-DR	52412	5800

coverage for the observed phenomenon, in a shorter simulation time interval. However, by the even rate curve, the baseline scheme has lower ability to advance the committed simulation time interval vs wall-clock time. Hence, even though the schemes exploiting space uncertainty will require to commit more simulated time for providing the requested coverage, they can be likely faster in terms of wall-clock time due to the higher event rate. This is exactly the result shown in Table 1, where, for each configuration, we report the mount of simulated time required for the requested coverage in the simulation model, and the best case (vs the checkpoint interval) wall-clock time requested for simulating that virtual time interval. By the results, BC-DR requires the lowest wall-clock time, instead the baseline requires the lowest simulation time interval. Hence, although the event rate of BC-DR is about 40% higher than that of the baseline, in terms of wall-clock time the gain is reduced to about 20% just because BC-DR needs to simulate a longer virtual time interval. A similar consideration can be made for RINAC, since the simulated time interval requested is even longer (since there is less control on the bias of the achieved statistics). Hence, even though the event rate of RINAC is similar to the one of BC-DR, this latter scheme is about 9% faster in terms of wall-clock time, thus revealing its effectiveness.

References

- [1] Tcl/tk project.
- [2] R. Beraldi and L. Nigro. A Time Warp based on temporal uncertainty. *Transactions of the Society for Modeling And Simulation*, 18(2):60–72, 2001.
- [3] D. Cucuzzo, S. D'Alessio, F. Quaglia, and P. Romano. A lightweight heuristic-based mechanism for collecting committed consistent global states in optimistic simulation. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 227–234, 2007.
- [4] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.
- [5] R. M. Fujimoto. Exploiting temporal uncertainty in parallel and distributed simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, pages 46–53. IEEE Computer Society, May 1999.
- [6] T. Kiesling, J. Lüthi, and R. E. A. Khayari. Bias in parallel and distributed simulation systems. In *Winter Simulation Conference*, pages 384–393, 2005.
- [7] M. L. Loper and R. M. Fujimoto. Pre-sampling as an approach for exploiting temporal uncertainty. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*, pages 157–164. IEEE Computer Society, May 2000.
- [8] F. Quaglia and R. Beraldi. Space uncertain simulation events: some concepts and an application to optimistic synchronization. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation*, pages 181–188. IEEE Computer Society, 2004.
- [9] R. Toccaeli and F. Quaglia. DyMeLoR: Dynamic memory logger and restorer library for optimistic simulation objects with generic memory layout. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, pages 163–172. IEEE Computer Society, 2008.