ON THE USE OF RANDOM NUMBERS
IN ASYNCHRONOUS SIMULATION VIA ROLLBACK\textsuperscript{1}

John N. Tsitsiklis\textsuperscript{2}

ABSTRACT

We consider the asynchronous distributed simulation of a stochastic system using the rollback method and we show that the simulation can be incorrect, meaning that the sample path generated is not distributed according to the desired statistics, unless some precautions are taken. In particular, if part of the simulation is performed for a second time, due to a rollback, one should use the same random numbers that were used the first time.

\textsuperscript{1} Research supported by the National Science Foundation under grant ECS–8552419, with matching funds from IBM Inc. and Dupont Inc.

\textsuperscript{2} Room 35–214, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139.
1. INTRODUCTION.

Asynchronous simulation via rollback is a relatively recent method for simulating a discrete-event system, consisting of interconnected subsystems, using a set of asynchronous processors, each processor being in charge of simulating a particular subsystem [J]. This method has attracted a fair amount of attention (see e.g. [MWM] and the references therein) with a view towards applications in the simulation of queueing networks and other types of stochastic systems [M].

The essence of the rollback method is that each processor simulates its own subsystem as fast as it can, and communicates to other processors so that they can appropriately simulate the effects of one subsystem on the other. If a processor has simulated its own subsystem further in the future than it should (that is, if it has neglected some interactions from other subsystems) then part of the simulation is invalidated and is performed again (this is called a rollback), properly taking into account the neglected interactions. In the case of the simulation of stochastic systems, stochastic effects are simulated using random number generators. This raises the following question: when a rollback occurs and part of the simulation is done for the second time, should we use the same random numbers as the first time, or should we generate new random numbers? While ease of implementation might suggest the generation of new random numbers, we show that this option leads, in general, to incorrect results. That is, the sample path generated by the simulation algorithm need not possess the desired statistics.

The remainder of the paper is organized as follows. In Section 2, we provide a description of the simulation algorithm. In Section 3, we discuss how the generation of new random numbers leads to incorrect results. Finally, in Section 4, we provide an informal argument suggesting that the use of the old random numbers leads to correct results, and discuss some related issues.

2. THE ROLLBACK ALGORITHM.

We provide here a condensed summary of the rollback method, our main purpose being to establish the terminology for our subsequent discussion. For a more detailed and accurate description, the reader could consult [J] or [BT]. Furthermore, in order to simplify the presentation, it is assumed that the system being simulated evolves in discrete time. Nevertheless, the same arguments can be applied to the case of continuous-time systems.

We refer to the system being simulated as the physical system, as opposed to the computing system, which consists of the processors performing the simulation. The physical system consists of \( N \) subsystems, denoted by \( S_1, \ldots, S_N \), and operates for a finite number \( T \) of discrete-time units. We let \( t \in \{0, 1, \ldots, T\} \) be a time variable associated with the physical system, to be referred to as the physical time. With each subsystem \( S_i \), we associate a state variable whose value at physical time \( t \) is denoted by \( x_i(t) \). Each subsystem can generate interactions which can affect the state of the other subsystems at subsequent physical times. We let \( z_{ij}(t) \) be a variable describing the nature of an interaction generated at subsystem \( S_i \) at time \( t \), that will affect subsystem \( S_j \). We
assume that the interaction variables can take a null value, denoted by \( \pi \), which stands for absence of interaction. Furthermore, without loss of generality, we assume that \( z_{ij}(t) \) affects directly the state \( x_j \) at time \( t + 1 \). We are thus led to the following model of the physical system:

\[
x_i(t) = f_i\left(x_i(t - 1),\{z_{ji}(t - 1) \mid j \neq i\}, w_i(t), t\right), \quad t \geq 1, \tag{1}
\]

\[
z_{ij}(t) = g_{ij}(x_i(t), w_i(t), t), \quad t \geq 0. \tag{2}
\]

Here, the \( f_i \)'s and the \( g_{ij} \)'s are functions describing the dynamics of the subsystems and the mechanisms that generate interactions. The variable \( w_i(t) \) is a random variable, meant to capture the stochastic aspects of the evolution of \( x_i(t) \). We assume that the random variables \( \{w_i(t) \mid i = 1, \ldots, N; t = 0,1,\ldots,T\} \) are independent and with prescribed distributions. (This independence assumption is more or less necessary in practical implementations in which the random variables are drawn using random number generators. Even if a natural description of the system violates such an independence assumption, the dynamical equations are usually reformulated and the random variables redefined so as to enforce the independence assumption.) We assume that initial conditions \( x_i(0) \) are provided for each subsystem and that they are deterministic (as opposed to random).

We continue with the description of the computing system. We assume that there are \( N \) processors \( P_1, \ldots, P_N \), each one being responsible for the simulation of a corresponding subsystem. Each processor \( P_i \) maintains a local clock (often called local virtual time, or LVT for short); its value will be denoted by \( r_i \). Each processor simulates its own subsystem as fast as it can and the value of \( r_i \) indicates that the values of \( x_i(0), x_i(1), \ldots, x_i(r_i) \) have been simulated and have not been invalidated. A typical simulation step is as follows. Assume that \( r_i = t \). Processor \( P_i \) draws a value of the random variable \( w_i(t + 1) \) (using a random number generator) according to the prescribed distribution. It then computes \( x_i(t + 1) \) and \( z_{ij}(t + 1) \), using Eqs. (1) and (2), transmits the value of \( z_{ij}(t + 1) \) to all processors \( P_j \) for which \( z_{ij}(t + 1) \neq \pi \), and increments \( r_i \) to \( t + 1 \). Notice that Eq. (1) requires knowledge of the interactions \( z_{ij}(t) \) emanating from the other subsystems. Processor \( P_i \) looks into a record of received messages for the values of the \( z_{ij}(t) \)'s. For any \( j \) for which no such message is found, the null value \( \pi \) is assumed.

Suppose now that processor \( P_i \) receives a message with a value of \( z_{ij}(t) \), where \( t < r_i \). This message invalidates the simulated values of \( x_i(t + 1), \ldots, x_i(r_i) \), which have to be simulated anew. Accordingly, processor \( P_i \) assigns the value \( t \) to its LVT. Furthermore, processor \( P_i \) sends antimessages to cancel any transmitted messages that were based on the invalidated values. The behavior of any processor that receives an antimessage canceling an earlier received message is similar: that is, all computations that depend on the value of the canceled message are invalidated and any message that was based on the invalidated computations is canceled by further antimessages.
3. THE POSSIBILITY FOR INCORRECT RESULTS.

As discussed in the previous section, rollbacks may force a processor to compute a value of \( x_i(t) \) several times, for the same value of \( t \). Let us assume that each computation of \( x_i(t) \) makes use of a different, independently drawn, sample of the random variable \( w_i(t) \). We indicate a mechanism that can lead to an incorrect simulation.

Let us consider an asynchronous simulation for which we can be certain that a fixed processor \( P_i \) will roll back at most once and, consequently, the value of a random variable \( w_i(t) \) will be sampled once or twice, depending on whether a rollback occurs or not. Let us assume that the prescribed distribution for \( w_i(t) \) is such that we have \( w_i(t) \in \{0,1\} \), with the two values being equally likely. Let \( w \) be the value at the first drawing of \( w_i(t) \) and let \( w' \) be the value at the second drawing, if a rollback occurs. Let \( \chi \) be the indicator function of the event that a rollback occurs. (That is, \( \chi = 1 \) if a rollback occurs, and \( \chi = 0 \) otherwise.) Since \( w \) and \( w' \) are generated according to the prescribed distribution, we have

\[
\Pr(w = 0) = \frac{1}{2},
\]

\[
\Pr(w' = 0 | \chi = 1, w = j) = \frac{1}{2}, \quad \forall j \in \{0,1\}.
\]

The value \( v \) finally used in the simulation of \( w_i(t) \) is equal to \( w \) if \( \chi = 0 \), and equal to \( w' \) if \( \chi = 1 \). Our simulation will be correct, meaning that the finally accepted value \( v \) of the random variable \( w_i(t) \) has the prescribed distribution, if and only if \( \Pr(v = 0) = \Pr(v = 1) = 1/2 \). We have

\[
\Pr(v = 1) = \Pr(w = 1, \chi = 0) + \Pr(w' = 1, \chi = 1)
\]

\[
= \Pr(w = 1) \cdot \Pr(\chi = 0 | w = 1) + \Pr(w' = 1 | \chi = 1) \cdot \Pr(\chi = 1)
\]

\[
= \frac{1}{2} \left( \Pr(\chi = 0 | w = 1) + \Pr(\chi = 1) \right)
\]

\[
= \frac{1}{2} \left( 1 - \Pr(\chi = 1 | w = 1) + \Pr(\chi = 1) \right).
\]

We therefore see that \( \Pr(v = 1) \) is equal to the desired value of \( 1/2 \) if and only if \( \Pr(\chi = 1 | w = 1) = \Pr(\chi = 1) \). Equivalently, if and only if the value \( w \) obtained at the first drawing does not affect the probability of having a rollback. One may envisage, however, complex sequences of events whereby the value \( w \) of \( w_i(t) \) at the first drawing determines whether certain messages will be sent by processor \( P_i \), thus affecting the traffic conditions in the interconnection network of the computing system, and indirectly influencing the probability that a rollback necessitates the regeneration of \( w_i(t) \). We provide below a detailed example which establishes our claim that the simulation has no correctness guarantees. Furthermore, our example shows that the above described pathological behavior can occur even if it is assumed that messages and anti-messages traveling from the same origin to the same destination are received in they order that they are sent.

Example
We consider a physical system consisting of three subsystems and which is to be simulated over the time interval \( \{0, 1, 2, 3\} \). To avoid a cumbersome presentation, we do not provide explicit formulas for the functions \( f_i \) and \( g_{ij} \) but we only describe their properties. It is left to the reader to verify that these functions can be easily chosen to have the properties that follow:

(i) The function \( g_{12} \) is such that \( z_{12}(0) = \pi \).

(ii) The function \( f_2 \) is such that \( x_2(1) = 0 \) if \( z_{12}(0) = \pi \), and \( x_2(1) = 1 \) if \( z_{12}(0) \neq \pi \).

(iii) The function \( g_{23} \) is such that \( z_{23}(1) = \pi \) if and only if \( x_1(1) = 1 \). As a consequence, we see that the interaction \( z_{23}(1) \) is nonnull if and only if the interaction \( z_{12}(0) \) is null.

(iv) There is a random variable \( w_{32}(2) \) that takes the values 0 and 1 with equal probabilities, and that determines whether the interaction \( z_{32}(2) \) is null or not. In particular, \( z_{32}(2) = \pi \) if and only if \( w_{32}(2) = 0 \) or \( z_{23}(1) \neq \pi \).

We assume that the above mentioned interactions are the only ones that can have nonnull values. The structure of these interactions is illustrated in Fig. 1.

In a correct simulation, we have \( z_{12} \neq \pi \), \( z_{23}(1) = 1 \), \( z_{32}(1) = \pi \), the random variable \( w_{32}(2) \) is drawn, and its value determines whether \( z_{32}(3) \) is null or not. In an asynchronous simulation there is a possibility of a rollback at processor \( P_3 \) that can lead to a redrawing of the random variable \( w_{32}(2) \). Let us consider the asynchronous simulation under two scenarios, corresponding to different outcomes in the first drawing of \( w_{32}(2) \).

In the first scenario, shown in Fig. 2, processor \( P_3 \) draws the value \( w_{32}(2) = 0 \). Accordingly, \( z_{32}(2) = \pi \) and no message is sent from \( P_2 \) to \( P_3 \). Furthermore, a message \( \alpha \) is sent from \( P_1 \) to \( P_2 \) with the value of \( z_{12}(0) \). This message reaches \( P_2 \) just before the variables \( z_2(1) \) and \( z_{23}(1) \) are simulated by \( P_2 \). Therefore, \( z_{23}(1) \) is simulated based on correct information, the value \( z_{23}(1) = \pi \) is obtained, and no message with the value of \( z_{32}(1) \) is sent.

In the second scenario, shown in Fig. 3, processor \( P_3 \) draws the value \( w_{32}(2) = 1 \). Accordingly, \( z_{32}(3) \neq \pi \) and a message \( \beta \) is sent from \( P_3 \) to \( P_2 \). As in the previous scenario, a message \( \alpha \) is also sent from \( P_1 \) to \( P_2 \) with the value of \( z_{12}(0) \). Suppose that the message \( \beta \) reaches processor \( P_1 \) just before message \( \alpha \). As both of these messages have to get into an "input queue" or "input buffer" for processor \( P_2 \), it is reasonable to assume that the reception of \( \beta \) can delay the reception of \( \alpha \). This causes the message \( \alpha \) to be received after the variables \( z_2(1) \) and \( z_{23}(1) \) are simulated. Thus, an incorrect nonnull value for \( z_{23}(1) \) is obtained. A corresponding message \( \gamma \) is sent from \( P_2 \) to \( P_3 \). When the message \( \gamma \) is received by \( P_3 \), a rollback occurs and an antimessage \( \beta' \) is sent to cancel the message \( \beta \). In the meantime, message \( \alpha \) reaches \( P_2 \), a rollback occurs at \( P_2 \), the value of \( z_{23}(1) \) is simulated correctly, and an antimessage \( \gamma' \) is sent to cancel the message \( \gamma \). Upon reception of \( \gamma' \), processor \( P_3 \) suffers one more rollback and draws the value of \( w_{32}(2) \) once more.

It is seen that the final value of \( w_{32}(2) \) is equal to 1 if and only if the value 1 is obtained at the first drawing and also in the last drawing. We are therefore simulating the system as if \( \Pr(w_3(2) = 1) = 1/4 \), which is incorrect. The root of the problem is that the value of \( w_{32}(2) \) at the
first drawing affects the probability that a rollback occurs and indirectly affects the probability that 
this random variable is redrawn. In this example, we have used an assumption that when there are 
more than one messages traveling towards the same destination, each one of these messages has 
the potential of delaying the reception of the others, which seems to be realistic from a practical 
point of view.

3. DISCUSSION.

We have seen that if random variables are regenerated each time a rollback occurs, the simulation 
can generate statistics different than the desired ones. We note that our arguments pertain to other 
contexts in which rollback and randomization are present. For example, rollback can be employed 
as a general purpose “synchronizer”, that is, as a protocol for executing synchronous algorithms in 
an inherently asynchronous computing system [A], [AS]. In fact, Eqs. (1) and (2) can be viewed as a 
general description of a synchronous algorithm, and the simulation of Eqs. (1)–(2) via the rollback 
method can be viewed as a “synchronizer”. If the synchronous algorithm involves randomization, 
the arguments of Section 3 apply verbatim.

What can be done to ensure the generation of the correct statistics? It is seems that the only 
alternative is to avoid redrawing new values $w_i(t)$ after each rollback. This implies that each 
processor must store in its memory the value of each $w_i(t)$, just in case a rollback is to occur later. 
We argue informally that such a strategy will lead to sample paths with the correct statistics. 
Indeed, if the value of each $w_i(t)$ is fixed at the value obtained at the first and only drawing of that 
random variable, the situation is mathematically equivalent to having drawn values for all of the 
random variables $w_i(t)$ before the simulation starts and then reading these values from the memory 
whenever they are needed. If the values of the variables $w_i(t)$ are drawn before the simulation starts, 
we are essentially dealing with the case of a deterministic simulation which employs some exogenous 
variables $w_i(t)$. Given that the rollback algorithm is correct for the simulation of deterministic 
systems, the trajectory being simulated will obey Eqs. (1) and (2), with the values of $w_i(t)$ having 
been sampled according to the desired statistics, which is our objective.

Unfortunately, the above described approach has certain drawbacks because of a potentially large 
increase in the memory requirements of the algorithm. Some partial remedies are the following:

(a) While the asynchronous simulation is being carried out, one can sometimes guarantee that none 
of the LVTs will ever drop below a certain value $\tau$. (This can be verified in a distributed manner 
by using the snapshot algorithm of [CL]; see [G] and [S] for further discussion of this point.) In 
such a case the processors are allowed to delete from their memory the values of $w_i(t)$, $t < \tau$, 
since they will never be required in the future.

(b) Instead of storing the value of $w_i(t)$, a processor could store the seed $s_i(t)$ that was fed to a 
random number generator in order to generate $w_i(t)$. If the value of $w_i(t)$ is needed again (due 
to a rollback) then a processor only needs to use the same seed $s_i(t)$. If $s_i(t)$ is a simple function
of \( t \), then processor \( P_i \) does not need to store the value of \( s_i(t) \) but can compute it on demand.

We now comment on the simulation of continuous–time discrete–event systems, which is the one that is of most interest in practice [M]. Since discrete–time systems can be viewed as special cases of continuous–time systems, the problems identified in Section 3 persist. On the other hand, the physical times at which events are to occur are not known in advance and the fix we described earlier is not applicable. We have, however, the following option. Suppose that the dynamics of the physical system have been formulated so that the statistics of the random variable \( w_i(t) \) corresponding to the \( k \)th event at subsystem \( S_i \) has a prescribed distribution depending only on \( i \) and \( k \). We can then generate random variables \( w^1_i, w^2_i, \ldots \), and the value \( w^k_i \) will be the one to be used for the simulation of the \( k \)th event at subsystem \( S_i \), no matter how many times the \( k \)th event has to be simulated (due to rollbacks) and even if different simulations of the \( k \)th event correspond to different physical times.

As a closing remark, if Eqs. (1) and (2) have a special structure, or if something more is known about the way that the asynchronous simulation is implemented, it is conceivable that the simulation is guaranteed to produce sample paths with the correct statistics even if no precautions are taken. This is issue is currently under study.

REFERENCES.


The physical system being simulated. The interaction $z_{23}(t)$ is null if and only if $z_{12}(0)$ is nonnull. The interaction $z_{32}(1)$ is nonnull if and only if $w_3(2) = 1$ and $z_{23}(1)$ is null.
The progress of the simulation, in real time, if the random sample of $w_3(2)$ has the value 0.

Figure 2

The progress of the simulation, in real time, if the first random sample of $w_3(2)$ has the value 1.

Figure 3