# TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER

*Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), the University of Maryland, Harvard University, and Industry*

# An Analysis of Rollback-Based Simulation

*by B. Lubachevsky, A. Shwartz and A. Weiss*

TR 91-37

# AN ANALYSIS OF ROLLBACK–BASED SIMULATION

by

**Boris Lubachevsky**
**AT&T Bell Laboratories**
**Murray Hill, NJ**

**Adam Shwartz***
**Electrical Engineering,**
**Technion—IIT, Haifa, Israel**

**Alan Weiss**
**AT&T Bell Laboratories**
**Murray Hill, NJ**

## ABSTRACT

We present and analyze a general model of rollback in parallel processing. The analysis points out three possible modes where rollback may become excessive; we provide an example of each type. We identify the parameters which determine a stability, or efficiency region for the simulation. Our analysis suggests the possibility of a dangerous "phase-transition" from stability to instability in the parameter space. In particular, a rollback algorithm may work efficiently for a small system but become inefficient for a large system. Moreover, for a given system, it may work quickly for a while and then suddenly slow down.

On the positive side, we give a tunable algorithm, Filtered Rollback, that is designed to avoid the failure modes. Under appropriate assumptions, we provide a rigorous mathematical proof that Filtered Rollback is efficient, if implemented on a reasonably efficient multiprocessor. In particular: we show that the average time $\tau$ to complete the simulation of a system with $N$ nodes and $R$ events on a $p$-processor PRAM satisfies $\tau = O\left((R/p) + (R/N)\log p\right)$.
The analysis is based on recent work by the authors on Branching Random Walks with a barrier.

**Keywords**: Time Warp, Efficiency of algorithms, Discrete-event simulation, Branching Random Walks.

Submitted April 1990, and in revised form March 1991.

* The work of this author was performed in part while he was visiting the Department of Mathematics of Networks and Systems, Mathematical Sciences Research Center, AT&T Bell Laboratories and the Systems Research Center, University of Maryland, College Park. Their support and hospitality are gratefully acknowledged.

# 1. INTRODUCTION

## 1.1. PREAMBLE*

Computer scientists have been concerned recently with the efficiency of rollback-based simulation algorithms [MM, LL, LMS, MWM]. We connect this with the phenomenon of phase transition, or "0-1 laws", which have recently raised substantial interest among mathematicians (e.g. [Sh]). We propose a model (a branching random walk with barrier— b.r.w.w.b.) of rollback-based algorithms, and show that the model (and hence actual rollback algorithms) possesses a phase transition: in one region, the b.r.w.w.b. terminates with probability one, and in the other it lives forever with positive probability. The transition is identified through two parameters, $b$ and $h$, which are an aggregate representation of the concrete simulation parameters. The branching factor $b$ roughly represents the average number of nodes that receive antievents or antimessages due to a particular node rolling back. The large deviations rate $h$ of the random walk (defined in equation (3)) is a measure of the tail of the change in rollback size as one moves down the rollback tree. The critical relationship is:

$$\textit{if } b < e^h \textit{ then } P(b.r.w.w.b. \textit{ terminates}) = 1 \tag{1.a}$$

and the rollback-based simulation is efficient. Conversely,

$$\textit{if } b > e^h \textit{ then } P(b.r.w.w.b. \textit{ terminates}) < 1 \tag{1.b}$$

and the rollback-based simulation may become inefficient, since a single rollback tree may then last forever. Our result carries an important warning to a rollback simulationist: a rollback-based simulation algorithm may work efficiently for one set of parameters, and the same algorithm may work inefficiently for a different set of parameters. Moreover, if $b > e^h$ the algorithm may work efficiently for a while, then abruptly slow down when a rollback tree that does not fade out is generated (see an example in Section 3).

A rollback-based simulation algorithm is a complex endeavor and may become inefficient from a variety of causes; most often practitioners are concerned with memory allocation. However, our analysis shows that even if memory is handled efficiently, the algorithm may be inefficient because of an imbalance between the speed of propagation of erroneous events within the simulation and the speed of their cancellation. We describe this imbalance in terms of the relations (1).

---

* We begin with an overview of our results and point of view, relying on standard terminology and concepts from rollback-based simulation. We will provide detailed background, definitions, and problem formulation in subsequent sections.

Let us explain in more detail the meaning of $b$ and $h$. A simulation generates a flow of dependent events in different processing elements (PEs) of a parallel processor. These flows form trees; e.g., event $e_1$ processed by $\mathrm{PE}_1$ causes processing of events $e_{11}$ in $\mathrm{PE}_{11}$ and $e_{12}$ in $\mathrm{PE}_{12}$, event $e_{11}$ causes processing of events $e_{111}$ in $\mathrm{PE}_{111}$ and $e_{112}$ in $\mathrm{PE}_{112}$, event $e_{12}$ causes processing of events $e_{121}$ in $\mathrm{PE}_{121}$ and $e_{122}$ in $\mathrm{PE}_{122}$, and so on. In a rollback-based algorithm errors are carried by this flow and erroneous simulations spread from PE to PE, forming trees. For example, if $e_1$ is an incorrect event, then all the events in the causally dependent event tree $e_{11}$, $e_{12}$, $e_{111}$, $e_{112}$, $e_{121}$, $e_{122}$ are generally incorrect. We define $b$ to be the average branching factor of these trees, that is the expected number of events directly generated by a given event.

Suppose that processor $\mathrm{PE_i}$, at some time during the interval $[t_i, t_i + X]$, schedules an event $e$ at processor $\mathrm{PE_j}$. Later, $\mathrm{PE_i}$ determines that it has incorrectly simulated the time interval $[t_i, t_i + X]$. It will therefore roll back and inform the "neighboring" processors, including $\mathrm{PE_j}$ that, in particular, event $e$ is incorrect. If $\mathrm{PE_j}$ has already processed event $e$ then some time interval $[t_j, t_j + Y]$ simulated by $\mathrm{PE_j}$ is incorrect. The larger $X$ is and the slower the information about incorrect events propagates, the larger we expect $Y$ to be. With the definition $W = Y - X$, $h$ is now defined as the large deviations rate of the random variable $W$ (see Section 2, Equation (3)).

A novel observation in our analysis is that the appropriate quantity to observe when studying rollback phenomena is $h$, rather than the mean of $W$, denoted $E(W)$. A "first order" analysis using $E(W)$ may be misleading: indeed, denote by $q$ the probability that $W > 0$, i.e., that a second-generation rollback is larger than the original. Then the probability that no second generation rollback is larger than the first rollback is $(1-q)^b$, from which one might guess that this wave of rollback is guaranteed to terminate whenever $(1-q)^b > 1/2$. This argument applied to a binary tree ($b = 2$) gives a threshold value of $q$ as $q_{wrong} \approx 0.3$. If, for example, $W$ takes only the values $\pm 1$, then the common requirement $EW < 0$ holds whenever $q < 1/2$, which is even weaker than $q < q_{wrong}$. Both of these are incorrect, and condition (1) turns out to be substantially more stringent (see Section 3.4); the true stability threshold given by (1) is $q_{right} = 0.067$! The mathematical model is detailed in Section 2.

Our analysis reveals the surprising *viability* of rollbacks, which is stronger than anticipated by other researchers. For example, if $q_{right} < q < q_{wrong}$, the rollback might temporarily (for the first few generations) or locally (at a fixed node) appear to decrease, but Theorem 1 states that ultimately the rollback may grow and live forever and its size may become unbounded. As discussed in Sections 2.2 and 6.2, our analysis provides an upper bound on the overhead of rollbacks, which is tight for some types of systems. For such systems an algorithm may become inefficient

for a sufficiently large simulation size, while appearing efficient for a small simulation size. For such systems, predicting the efficiency of a large simulation through small pilot runs might be misleading. The way in which such instability occurs is also understood from our model, and will be detailed in Section 3.

The identification of $b$ and $h$ as the fundamental parameters of rollback allows us to classify the failure modes of rollback. We find that exactly three modes exist. The viability of rollbacks in this classification corresponds to a heretofore unknown mode, *wildfire cascading.* We give examples of each of the failure modes in Section 3.

Rollback simulationists have always felt that, in order for the simulation to be efficient, event cancellation must be faster than the forward simulation flow. The quantities $b$ and $h$ may be interpreted as encoding these two speeds, and (1) thus becomes an exact formulation of this "folk theorem." In an implementation, speeding up the cancellation means slowing down the forward simulation (as in [Fb]), because during the forward phase, the algorithm spends additional efforts to facilitate possible subsequent cancellation (by precomputing pointers, lists etc.). Such an algorithm proceeds by "walking" ahead while building a "highway" for "wheeling" back.

However this is not the only way to interpret the parameters $b$ and $h$, nor is it the only way to ensure algorithm efficiency. In Section 4 we describe in detail a different strategy called *Filtered Rollback* [LSWa]. Filtered Rollback puts more emphasis on selectively slowing down those forward computations that are more likely to carry the errors, rather than uniformly slowing down all computations. In this way efficiency may coexist with a slow cancellation (although any inexpensive means of speeding the cancellation would further increase efficiency). The method can be considered as a modification of the previously proposed *Bounded Lag* algorithm [Lb] extended to allow the possibility of (and the recovery from) overestimations of internode propagation delays.

In section 5 we demonstrate the efficiency of the Filtered Rollback algorithm. Specifically, suppose that the total number of events we wish to simulate is $R$, the number of processors we use is $p$, and the number of nodes where events occur is $N$. We assume that $N$ is a correct measure of the inherent concurrency of the system being simulated (as detailed below, this means that load imbalance does not grow with $N$). We suppose that $R \geq N \geq p$. Then under the technical assumptions of Section 5, we show that the average time $\tau$ to complete the simulation satisfies

$$\tau = O\left(\frac{R}{p} + \frac{R}{N}\log p\right). \tag{2}$$

For example, if on average each node has a constant number of events to simulate, and the number of processors grows linearly with the number of nodes, then we find $\tau = O(\log N)$. If the number of

events to simulate grows as the square of the number of nodes, and we again have $O(N)$ processors, then we find $\tau = O(N \log N)$.

The proof that Filtered Rollback is efficient holds for systems that are *large, loosely coupled* and *busy*. The largeness assumption is basically that there are a lot of events to simulate, and a lot of processors available to do the simulation. "Efficiency" is an asymptotic term as in (2): as the problem size scales, the simulation does not slow unduly. The loosely coupled nature of the system is used in a technical point of the proof, but essentially means that a distributed algorithm has a chance of being efficient because not all the time is spent coordinating processors ... each only communicates with a rather restricted set of the system. The "busy-ness" assumption means that a good fraction of the processors are busy at all times. This again seems necessary for a distributed simulation to have a chance of working, and is also an assumption on the efficiency of the multiprocessor hosting the simulation. Thus, load imbalance does not grow with the size of the simulation. Our proof of efficiency involves showing that not too much effort is wasted on rollbacks, or on simulating unnecessary events. The methods we use may be applicable in more general settings than we claim, but we wanted to provide a specific and complete analysis of one distributed algorithm before exploring the scope and full power of our methods.

## 1.2 SYSTEMS UNDER CONSIDERATION

In this paper the task of simulation is understood as the construction of the time history of a simulated system or model*. In a discrete-event simulation, the history consists of instantaneous *events*, each event $e$ represents a change of state of the system or its component. In the computer simulation, an event $e$ is represented by a record $e = (time,\ contents)$, where $time = time(e)$ is the time of the change and $contents = contents(e)$ is a specification of the change the event represents. When the simulated system consists of many components, and an event (state change) occurring at a component typically does not immediately affect many other components, one might look for a distributed parallel simulation method wherein different components are assigned to different processing elements (PEs) of a parallel computer. We will assume that each PE is permanently assigned to simulate its own component. This is not a restriction at this stage of the discussion, because no method of division of the whole system into the components has been specified.

Since a typical simulation problem we have in mind is a network simulation, the components

* When some statistical characteristics only, but not the history as a whole are needed, then techniques such as "change of measure" and other enhancements of Monte Carlo simulations can be usefully employed.

will be called *nodes*, although it is quite possible that a "node" identifies something that does not look like a network node: e.g, a space sector in the "billiards" simulation [AW,HB,Ld], a combination of several nodes of a network, an imaginary logical node in a data processing graph etc. We insist, however, on a notation such that each event $e$ occurs at only one node and the $contents(e)$ includes the node identification, $node(e)$. Processing event $e$, besides changing the state of $node(e)$, may cause the scheduling of some events or cancellation of some previously scheduled events for $node(e)$ or other nodes. We assume that each node $i$ (or, speaking more accurately, the PE that carries node $i$) maintains its local time $T(i)$ and a pool $\Pi_i = \Pi_i(T(i))$ of *tentative future events* so that for any $e \in \Pi_i(T(i))$ we have $time(e) \geq T(i)$. Since this is a discrete-event simulation, the local time $T(i)$ coincides with the time of the earliest event in $\Pi_i(T(i))$.

We conclude this general picture with one important but usually not well spelled out detail on the presence of *randomness* in a simulation. Randomness may be present in a *model* we simulate on a computer. However, once the model is settled and the computer program is written, randomness disappears. Nothing is (or at least is supposed to be) random in an execution of a reproducible computer program. Pseudo-random number generators are deterministic algorithms and the simulated history is uniquely determined by the initial conditions. Thus the simulation in this paper is treated as completely deterministic although its analysis uses probabilistic arguments.

### 1.3 TO ROLL OR NOT TO ROLL

Two different ways of simulating a discrete event system have been proposed. Jefferson [J] christened them "conservative" and "optimistic"; the terms have been adopted in the simulation literature [Fd].

In conservative algorithms, "... what's done is done"* : when we begin processing event $e$ we are supposed to be absolutely certain $e$ exists in the system history. In general, the price of 100% certainty that a scheduled event is "correct" is serialization of simulation: we can only be sure that it is correct to process a scheduled event $e$ if $time(e)$ is the minimum of the times of all presently scheduled events. In some systems, this serialization can be overcome using "lookahead" [Fa], minimum propagation delays [Lb], or "future event lists" [N]. This serialization is not felt, of course, if the method of simulation is already serial. That is why the division into the two simulation modes became apparent only with the advent of parallel computers.

In the optimistic approach, provisions are made to undo simulated events if evidence is obtained that these events are wrong. A simple optimistic method is *checkpointing*: once in a while the state

* Shakespeare, *Macbeth*, Act III Scene 2.

of the simulation is saved, forming a point of return, or checkpoint. Whenever an error is detected, the state of the entire system is restored to the nearest safely passed checkpoint. Checkpointing is convenient when the errors are rare [Ld].

If the errors are frequent, then *relaxation* is a more natural optimistic method. *Asynchronous* relaxation (*rollback*, or *Time-Warp*) algorithms were introduced by Jefferson [J]. Under such algorithms, each node of the simulation takes care of its own checkpointing, and corrects errors independently of the other nodes. Each node has its own idea of what time it is (being simulated), namely, *local time* $T(i)$ is the time of the event node $i$ is currently processing. As a result of this asynchrony, it is possible that node $i$ schedules an event $e$ for execution in the past of node $j$. Insertion of $e$ into the pool $\Pi_j(T(j))$ of tentative events for node j with $time(e) < T(j)$ is considered as evidence of an error made by node $j$ and all or part of the processing that was done during the local time between the time $time(e)$ and current local time $T(j)$ must be undone. This "undoing" is called rolling back and we will measure the amount of this undoing by the length of the interval [time(e), $T(j)$] which we will call the (amount of) rollback. A difficulty in correcting and analyzing the damage done by incorrect simulations is in the fact that in the course of incorrectly simulating the interval [time(e), $T(j)$] node $j$ might have inserted events, now known to be incorrect, into the pools of its neighbors.

The following method of error correcting was introduced by Jefferson [J]. While undoing the interval [time(e), $T(j)$], the node inserts "antievents" into the neighboring nodes. An "antievent" is a record that locates its corresponding "positive" event uniquely and contains an indication that the "positive" event is wrong. The antievents may lead to further antievents in a sort of chain reaction. This process is analyzed in detail in this paper. Usually, the discussion of rollback is couched in terms of "messages" and "antimessages", as in the original Time Warp algorithm [J]. We hope that the reader will bear with our terminology which we believe to be natural for a larger class of systems.

The rollback discussed above is "aggressive" in the sense that whenever node $j$ receives an event $e$ with $time(e) < T(j)$, it rolls back to time $time(e)$. However, it may not be necessary to roll back. A rollback scheme that first tries to incorporate the newly scheduled event *without rolling back* is called a "lazy" scheme. In an aggressive scheme, if node $i$ has several antievents to send to node $j$, it clearly suffices to send the one with the earliest time. This is not the case in a lazy scheme. Thus under a lazy scheme communication overhead is larger, and moreover since rollback is delayed, differences in local times tend to be larger. On the other hand, a rollback resulting from the same antievent in the same state will generally be smaller in a lazy scheme. It is thus not

obvious which scheme is more efficient, and since they are clearly different, our analysis is restricted to one scheme — aggressive cancellation.

To our knowledge, all reported experiments on rollback algorithms have been conducted on MIMD (Multiple Instruction Multiple Data) machines. The lack of synchrony in these algorithms makes them ill-suited for SIMD (Single Instruction Multiple Data) machines. However, synchronous relaxation is well suited for SIMD computations. We are aware of only two papers dealing with synchronous relaxation. Chandy and Sherman [CS] discuss the general application of relaxation and fixed-point calculation to discrete-event simulations. Greenberg, Lubachevsky and Mitrani [GLM] discuss a synchronous relaxation algorithm for a network with a special structure. The algorithm proposed in this paper is applicable to fairly general networks.

Since relaxation has potential problems, it might seem reasonable to use a conservative scheme, such as Bounded Lag. The serialization common to all conservative algorithms here takes the following forms: Either a simulationist may find it difficult or impossible to provide a guaranteed positive lower bound on the delay of propagating causality (see [HB, Lc]), or he may provide a uselessly low bound. Causality propagates from node $i$ to node $j$ when an event processed at node $i$ schedules an event at node $j$. Without a bound the algorithm cannot be parallelized. In the common toy where 5 steel balls are suspended from a frame all just barely touching, when an outside ball is swung back and allowed to fall against the others, the far ball pops out with no appreciable delay. This is an example of zero propagation delay, and has dire consequences for a conservative algorithm. A related, but somewhat different issue, is the efficiency of the algorithm. Even if a positive lower bound on delay is provided, it may be so small that an implementation of the scheme would have events processed in serial order, giving very poor performance. All processors except for the one with globally smallest local time would be idle.

"Safety features" such as "Moving Time Window rollback" [SS] that prevent excessive difference in local times can be added to a rollback algorithm. Filtered Rollback is another amalgam which provides a safety feature; it is described in Section 4.

## 2. A MODEL OF ROLLBACK CASCADING

We begin with a description of the mathematical model we use as the basis of our analysis. This order of presentation is chosen so that we can substantiate our analysis of failure modes of rollback.

### 2.1 THE MODEL

We consider each rollback as a node of a tree, which may "branch", i.e. induce more rollbacks. The nodes are labeled $(n, j)$, where $n$ is the generation number (or depth), and $j$ identifies the node.

Our mathematical model is a branching random walk with a barrier (b.r.w.w.b). We start with a standard branching (Galton-Watson) process $k(n)$ [AH]: given the integer-valued, nonnegative i.i.d. random variables $\{y_{n,j}\ ,\ j = 1, 2, \ldots,\ n = 0, 1, \ldots\}$, we define

$$k(0) = 1\ , \qquad k(n+1) = \sum_{j=1}^{k(n)} y_{n,j}\ ,\ n = 0, 1, \ldots$$

so that $k(n)$ represent the number of $n^{\text{th}}$ generation offspring of an initial ($0^{\text{th}}$) generation ancestor, and $y_{n,j}$ is the number of children of $(n, j)$.

We define a random walk on this branching process by assigning a size (location) to each offspring. Start with $x_0$, the initial size, which represents the size of the initial rollback and let $x(n, j)$ denote the size of node $(n, j)$. Let $\{W_{n,j}\ ,\ j = 1, 2, \ldots,\ n = 1, 2, \ldots\}$ be i.i.d. random variables, independent of $\{y_{n,j}\ ,\ j = 1, 2, \ldots,\ n = 0, 1, \ldots\}$, where $W_{n,j}$ represents the change in rollback size of node $(n, j)$ from that of its immediate ancestor. We assume that on the average, the rollback of a descendant is smaller than that of his parent, but there is a positive probability it will be bigger:

A1) $EW_{i,j} < 0$,

A2) $P(W_{i,j} > 0) > 0$, and

A3) There is some $\alpha > 0$ such that $Ee^{sW_{i,j}} < \infty$ for $0 \leq s < \alpha$.

We do not expect assumption A3) to be burdensome; in fact, all random variables with exponentially decreasing tails satisfy it, such as Normal, Poisson, Binomial, and of course Exponential.

Finally, since a node with "negative rollback" will not induce further rollbacks, we introduce a barrier at 0. Any offspring $(n, j)$ with size $x(n, j) \leq 0$ is killed, along with all its offspring. Thus $(n+1, j)$ has size $x(n, i) + W_{n+1,j}$ whenever $(n, i)$ is the ancestor of $(n+1, j)$, provided the sum is positive; otherwise, this offspring does not exist. We denote by $z(n)$ the number of live offspring at the $n^{\text{th}}$ generation. Figure 1 depicts a rollback tree, and the development of rollback size along each branch. The process $x(n, m)$ is the model of rollback cascading we will use. It is nonnegative, and is positive only if all predecessors are positive. It encompasses both deterministic and random branching, has a barrier at zero and (as Theorem 1 demonstrates) has a rather complete analysis.

## 2.2 ROLLBACK VS. THE MATHEMATICAL MODEL

We now show that the branching random walk is an upper bounding model for the growth of rollback trees. We have to assume that the sizes of rollbacks from generation to generation are stochastically bounded, as a process, by a random walk with i.i.d. increments; see Section 5.1,

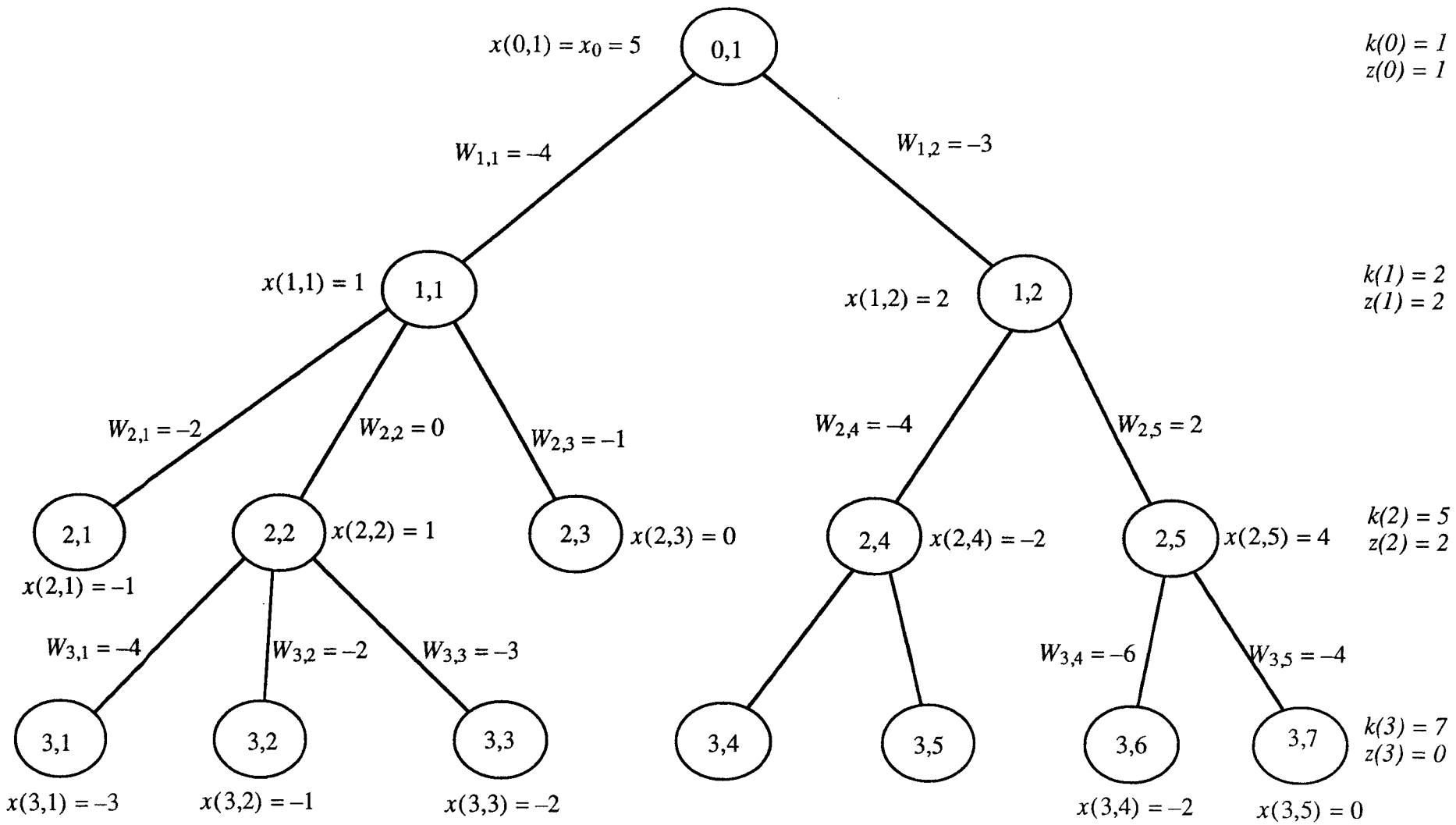


**Figure 1:** A Model of Rollback Cascading, a Branching Random Walk with a Barrier

Assumption **3**. That is, we are assuming that rollbacks are propagated in an easily estimable way, and we now want to bound the errors caused by the finiteness of the number of nodes in the underlying system, and the noninteracting nature of the rollbacks in the b.r.w.w.b. model. Let us examine the size of a rollback tree. When a rollback starts, it is because an event schedules another event in a different node with time smaller than the current local time. The size of the rollback, at this point, is irrelevant, and we model it as $x_0$ in the branching random walk. Next, we consider the rollback tree starting from this original rollback. The events in the simulation each schedule, on average, events in $b$ other nodes. Some of these events may actually be scheduled in the same node, and our model is an upper bound here because it assumes that all are scheduled in different nodes. (The rollback is worse if the nodes are all different, as then there is a larger population potentially scheduling antievents.) Later, when the tree is larger, several nodes may schedule antievents at the same node at the same cycle. The total number of involved nodes is again overestimated by the model, which never has such a coalescence. Furthermore, if two different rollback trees coalesce, only the "larger" rollback is effected, and the other tree does not continue; that is, the interference between rollback trees is overestimated. The stochastic model of the $W_{i,j}$ can be taken as an upper bound by simply taking the distribution of the $W_{i,j}$ as the largest among all distributions that actually appear in the simulation (in the sense of stochastic ordering; see Section 5.1, Assumption **3**). Thus we have seen that every error in the model is in the form of an overestimate on the size of a rollback tree. The upper bound is tight, though; see Section 3 and the discussion of the wildfire cascade.

In an actual simulation, we can expect the statistical assumptions of Section 2.1 to be violated. It is possible to handle more general models in much the same way as we did here, although we prefer to enforce the more restrictive assumptions, stressing the structure and intuition at the expense of generality. We conclude the section with a brief description of possible extensions and references. We can expect, for example, that a well-designed algorithm would have some stabilizing properties, so that the direct offspring of a node with a large rollback may experience fewer increases in rollback size than those of the direct offspring of a node with a small rollback. In terms of the mathematical model, this entails a dependence of the distribution of $W_{n+1,j}$ on the size $x(n,i)$ of the ancestor. The results of Azencott and Ruget [AR] may be applicable to this case. In particular, in a real simulation there may be a maximal rollback size (determined, for example, by the length of the simulated process or by memory limitations). The resulting mathematical model has an upper (reflecting) boundary, in addition to the lower absorbing boundary. The analysis of this case is under study and should pose no serious difficulty.

Dependence of the branching $\{y_{n,j}\}$ and walking $\{W_{n,j}\}$ can be handled as in Biggins [B]. In this case, and probably in the previous one as well, the results remain unchanged: there is still a phase transition, although it may be more difficult to compute its location.

### 2.3 STABILITY CONDITIONS

Since $z(n)$ is an upper bound on the number of nodes rolling back at the $n^{\text{th}}$ generation, it is natural to ask under what conditions it is finite, and moreover when $E\sum z(n) < \infty$. If the last inequality holds, then the rollback trees have a bounded effect, independently of the size of the system, so that we may be able to establish the efficiency of a rollback simulation.

To answer these questions, we need the notion of a large deviations rate. Define

$$h = \sup_{-\infty<\theta<\infty} \left\{-\log Ee^{\theta W_{1,1}}\right\} \tag{3}$$

and denote by $\theta^*$ the minimizer in (3), which is positive and is unique by convexity; see [LSWb, BLSW]. This rather technical definition is used in Theorem 1. It comes from Chernoff bounds on the random variables $W_{i,j}$; see [LSWb, BLSW]. The quantity $h$ is called sometimes the Cramer exponent, and sometimes the Chernoff rate. It may be understood in the context of Chernoff's theorem, in that it gives the asymptotic rate of occurrence of rare events. Note that if node $x(n,j)$ is alive, then $x(n,j) = x_0 + \sum_{m,i} W_{m,i}$, where the sum is over the indices of the $n$ ancestors of $(n,j)$, and that for a fixed $n$, the $\{x(n,j),\ j = 1,2,\ldots\}$ are identically distributed. For such sums of $n$ i.i.d. variables, Chernoff's theorem [Ch] states that

$$P\left(\sum W_{m,i} > 0\right) = e^{-nh+o(n)}.$$

This is a refinement of the Central Limit Theorem, which states that $P(\sum W_{m,i} > 0) = o(\frac{1}{\sqrt{n}})$. The following Theorem 1 is proved in [LSWb, BLSW]. It contains the basic result of the existence of a critical point, the dividing line in the phase transition between finiteness or unboundedness of our cascade model.

**Theorem 1.** *If $b < e^h$ then* $E\left(\sum_{n=0}^{\infty} z(n)\right) < \infty$ *and* $\lim_{n\to\infty} P(z(n) > 0) = 0$.

*Furthermore, if $b > e^h$ then* $E\left(\sum_{n=0}^{\infty} z(n)\right) = \infty$ *and* $P(\limsup_{n\to\infty} z(n) < \infty) < 1$.

In our use of this mathematical result we will want to be in the case where $be^{-h} < 1$, since this will imply that the rollback cascade is bounded, and even (as will be seen below) small.

### 2.4 SIZE OF ROLLBACK

In [LSWbThm. 2, BLSWThm. 2 (ii)], we show that in some sense, the largest rollback occurs in the initial rollback. Define $Y = \sup_{n,i} x(n,i)$.

**Theorem 2.** *If $b < e^h$ then $\lim_{x_0 \to \infty} \frac{Y}{x_0} = 1$ a.s. and there is a constant $C$ such that, with $\theta^*$ as defined below equation (3), $P(Y \geq t + x_0) \leq Ce^{-\theta^* t}$.*

This theorem shows that the initial rollback has essentially the largest rollback size in the entire tree it generates. We use it in technical points of our proof of the efficiency of Filtered Rollback. It might also be used in a study of memory management in rollback. For instance, if one can support the memory for an initial rollback, then one can support the entire tree's memory. For more detailed asymptotics on the size of rollback trees as a function of the initial rollback size, see [LSWb, BLSW].

## 3. CONSEQUENCES OF THE ANALYSIS

### 3.1 SOME CONSEQUENCES OF THEOREM 1.

We have seen the connection between the branching random walk with a barrier $x(n,m)$ and rollback cascading in Sections 2.1–2. Using this connection, we now apply Theorem 1 in order to identify the failure modes of rollback-based simulation.

| MODES OF ROLLBACK | | |
|---|---|---|
| | $b \leq 1$ | $b > 1$ |
| $E(W) \geq 0$ | Echo | Gushing Cascade |
| $E(W) < 0$ | Small | Wildfire Cascade |

Table 1. Modes of rollback

Below we discuss and illustrate these cases. Roughly speaking, an echo occurs when rollback grows bigger, leading to large overhead. This is usually the result of poor design, which leads to

$E(W) \geq 0$. Gushing cascade occurs when, in addition, $b > 1$ so that not only is rollback becoming larger, but the *number of nodes* involved in the rollback increases.

When $b \leq 1$ and $E(W) < 0$ the rollback is small in the sense that, whenever a rollback starts, it tends to decrease quickly in size, and the number of nodes involved decreases (if $b = 1$ it may increase slowly).

The most interesting (and heretofore unknown) case is wildfire. Wildfire occurs when the number of rollbacks increases in time. The fact that $E(W) < 0$ implies that, on the average, individual rollbacks decrease in size. However, their number tends to increase. The consequences of wildfire to the efficiency of the simulations depend on more careful analysis; see (1) and Section 3.4 below.

The following simple examples of the failure modes are intentionally naive and simple. It is clear that each may be cured in a variety of ways. In a large system, though, in an attempt to fix one problem, another may be exacerbated. For example, the "echo" problem of 3.2 might be alleviated by giving priority to the simulation of events, over the scheduling of antievents. This makes $E(W)$ smaller, but increases $b$. An echo may thus be transformed into a wildfire cascade.

We do not mean to imply that the situation is bleak. Indeed, we present a provably efficient algorithm, and it is certainly not the only such algorithm. Our hope is that the design of better algorithm will be facilitated by an understanding of the possible failure modes illustrated below.

### 3.2 ECHO

An echo is a cascade where the size of each rollback tends to grow, but the number of involved nodes does not increase. That this is possible is demonstrated by the following example.

Imagine a queuing system with two queues, A and B, and two types of customers, 1 and 2. Type 1 customers have preemptive priority over type 2 customers; that is, service of a type 2 customer is interrupted as soon as a type 1 customer requires service at the same facility. The service of an interrupted type 2 customer has to start afresh when it is resubmitted.

Upon completion of service at node A (respectively B), a type 1 customer enters queue B (respectively A). Type 2 customers leave the system after completing service at either queue. Type 1 customers take 1 unit of time to service, type 2 customers take 2 units of time. Initially there is an infinite number of type 2 customers in queue A as well as in queue B, and one type 1 customer in queue A.

In analyzing this system, we let (1) denote the processing of type 1 customer, and $(2_n)$ denote the processing of $n^{th}$ type 2 customer.

Applying the rules, we see that the actual system behaves as in Table 2. In the actual system, type 2 customers never complete service due to preemptions by the type 1 customer.

| Time | Node A Service | Node B Service |
|---|---|---|
| $[0,1)$ | $(1)$ | $(2_1)$ |
| $[1,2)$ | $(2_2)$ | $(1)$ (preempts $(2_1)$) |
| $[2,3)$ | $(1)$ (preempts $(2_2)$) | $(2_1)$ |
| ... | ... | ... |
| $[2n-1,2n)$ | $(2_2)$ | $(1)$ (preempts $(2_1)$) |
| $[2n,2n+1)$ | $(1)$ (preempts $(2_2)$) | $(2_1)$ |
| ... | ... | ... |

Table 2. The Simulated Queueing Network

Now suppose that the system is simulated by a rollback-based simulation, where

1. Node a hosts queue A, node b hosts queue B.
2. It takes one unit of physical time to process all types of events in the simulated system: the servicing of (1) or (2) customers, or the scheduling of antievents. Thus the "local clock" runs twice as fast when servicing a type 2 customer relative to servicing type 1 customer (compared with the actual system).

The simulation is described in Table 3. As in Table 2 we denote by $(x)$ the processing of customer $x$. We denote by $)2_n($ the "undoing" related to the n$^{\text{th}}$ type-2 customer, which includes rollback and scheduling antievents.

| physical | Node A Service | | Node B Service | |
|---|---|---|---|---|
| time | local time | activity | local time | activity |
| [0,1) | [0,1) | $(1)$ | [0,2) | $(2_1)$ |
| [1,2) | [1,3) | $(2_2)$ | 1 | $)2_1($ |
| [2,3) | [3,5) | $(2_3)$ | [1,2) | $(1)$ |
| [3,4) | 2 | $)2_3($ | [2,4) | $(2_4)$ |
| [4,5) | 2 | $)2_2($ | [4,6) | $(2_5)$ |
| [5,6) | [2,3) | $(1)$ | [6,8) | $(2_6)$ |
| ... | ... | ... | ... | ... |

Table 3. Simulation of the Queueing Network

In the simulation, since the processing of a type 2 event takes one unit of time, service of a type

2 customer is completed before it is preempted by the type 1 customer. This error is discovered wherever one node schedules an event to another — which is precisely wherever the type 1 customer completes service. Then, rollback (and preparation of antievents) takes place, so at that node, local time does not progress.

It is easy to see by induction that at physical time $n(n+1)/2 = 1 + 2 + .... + n$, the smallest local time of the simulation is $n$, which is also the simulated time of $A$ if $n$ is odd, or of $B$ if $n$ is even. The other of the two nodes is incorrectly advanced to simulated time $(3n-1)$ and has processed $n$ incorrect type-2 events. Thus, the rate of advancement of simulated time per unit of physical time goes in inverse proportion to the square root of the physical time, as the rollback amplitude rises in direct proportion to the square root of physical time.

In this example $b = 1$, since the number of nodes involved doesn't change. That is, each rollback-causing event and each antievent are sent to exactly one place. We can also see quite simply that $E(W) = 2$ . Each time a correct event is processed (event 1) the other node has to roll back two more time units than the last node did.

Notice that the assumption in 2 that "sending one antievent takes one unit of time" implies a straightforward programming mechanism which slides backward along the linear event list and produces and sends one antievent for each processed event subject to cancellation. Thus cancelling $m$ antievents takes time $\Omega(m)$. Recently, D. Jefferson pointed out (private communication) that this process can be substantially speeded up by using a more sophisticated event storage method, so that cancelling $m$ events *sent to a given recipient-node* takes only $O(\log m)$ time. Using such a method, $O(\log m)$ time is spent on sliding back the list till the earliest event to be cancelled and then time $O(1)$ is spend on sending *only one* antievent, the one with the earliest time. Under this algorithm the recipient "infers" that all later events must be cancelled. Our echo example however is still valid if the number of recipients is large. Even if this number is small, an echo can develop, which has a bounded, but practically large amplitude, because the sophisticated event storage methods outperform the straightforward linked-list only for a rather large $m$. See also the discussion in Section 6.

It is easy to generate an echo by "maliciously" manipulating processing rates for different types of simulated activities *during simulation.* An attractive and realistic feature of the example is that processing rates for the simulated activities are assigned *in advance* so that the echo is generated without outside intervention, just by virtue of the pattern of node interactions. Moreover, the example is stable in the sense that one may twiddle with all of the parameters in the problem and still be assured of generating an echo (the set of parameters that generate an echo has nonempty

interior, and the example presented here is on the boundary of the set).

### 3.3 A ZERO-EFFICIENCY SIMULATION

We now consider the borderline case of $E(W) = 0$ and show that a careless design may lead to zero efficiency. More precisely, we simulate a system with a finite number of events and finite number of nodes, but the simulation never terminates.

The system consists of 3 nodes, at the vertices of a triangle. At time 0 in the real system, the nodes send a "stop" instruction to their clockwise neighbor (node 1 $\rightarrow$ node 2, node 2 $\rightarrow$ node 3, and node 3 $\rightarrow$ node 1). The instructions arrive at time 1, and no further events are generated. If a node does not receive a "stop" instruction, it sends a "go" instruction to its clockwise neighbor, and any node receiving a "go" instruction passes it on at the next unit of time. Imagine that the simulating system acts exactly the same, with the exception that node 2, erroneously believing that node 1 sent no "stop" instruction, at time 2 sends a "go" instruction. Then node 2 receives the "stop" that it should have received at time 1, rolls back to local time 1, and sends an antievent to node 3. Meanwhile, node 3 sent a "go" instruction to node 1 at node 3's local time 3. At the next step, node 3 receives the antievent and passes it along to node 1, but not before node 1 has passed the "go" instruction to node 2. In this situation the antievent and instruction chase each other around the loop forever, with local time increasing at linear rate, and an infinite number of false events are processed with only a finite number of "inherent" events.

This example clearly has $E(W) = 0$ (in fact, $W = 0$), since the nodes roll back exactly the same number of time units at each rollback, and so $e^h = 1$. We also have $b = 1$, since the event and antievent are sent to exactly one node each time. The feature that distinguishes this example from "echoing" is that *all* the nodes in the simulator are involved in the rollback tree. This is clearly a borderline case, since $b = e^h$; we might have defined such a tree as an echo. To make $b > e^h$ in the example we should have an underlying topology where the number of involved nodes grows more quickly with time. See the "wildfire" example for such a topology.

Here, the underlying simulation is inherently unstable: rollbacks simply do not tend to die out. This sort of "dog chasing its tail" phenomenon is well known. In the next section we describe a new sort of cascading entitled Wildfire Cascading.

### 3.4 WILDFIRE CASCADE

It seems that this type of cascading was not suspected previously. Indeed, in her Ph.D. dissertation [Sa], B. Samadi indicated that it was well known that when $E(W) < 0$, there will be no extended cascading. The previous example can indeed be cured by somehow making $E(W) < 0$ in the

simulator; for example, by assuring that antievents travel, on average, more quickly than their corresponding events. Theorem 1 indicates, though, that cascading may occur even when $E(W) < 0$ unless the condition $b < e^h$ holds. We illustrate with two simple calculations below that wildfire cascading can start even in systems which appear to be rather stable. We then provide simulation results of a model of wildfire cascading in the shuffle exchange network, and compare those with the mathematical upper bound.

Consider a rollback tree which is a simple binary tree with $P(W = +1) = q$, $P(W = -1) = 1 - q$. This will have negative mean whenever $q < \frac{1}{2}$. However, a calculation of $be^{-h}$ gives $be^{-h} = 2b\sqrt{q(1-q)} = 4\sqrt{q(1-q)}$ so that the stability condition is

$$q < \frac{2-\sqrt{3}}{4} \approx 0.067 \ .$$

Theorem 1 states that this b.r.w.w.b. will have positive probability of living forever whenever $q > 0.067$. This means that one needs a rather strong negative drift (negative mean) in order for wildfire not to exist.

For another example, let $P(W = +1) = \frac{1}{2}$, $P(W = -K) = \frac{1}{2}$. In this example the drift is $(1 - K)/2$ which can be made as negative as we wish. We easily obtain

$$be^{-h} = K^{\frac{1}{K+1}}(1 + \frac{1}{K}).$$

But the right hand side of this equation is larger than 1 *for all finite K*. Thus, for all K there is a positive probability that the b.r.w.w.b. will live forever! The moral is that it may be difficult to control wildfire cascading.

One might argue that since the mathematical model involves, by definition, an infinite number of nodes, its relevance to practical simulations is tenuous. We now present an example of a system (in fact a practical network topology) which is finite, but exhibits the same qualitative behavior, and quantitative behavior that is very close to our infinite mathematical model.

This example is less satisfying than the echo or zero-efficiency examples since it does not portray a simulation, but rather a model for it. It is not difficult to construct an example for an infinite simulation system that will be identical to the mathematical model. The reason it is difficult to provide a more complete finite example is that wildfire can only exist on large simulators; small scale simulations with $E(W) < 0$ will not support wildfire. The example does show, however, that finiteness is not sufficient to preclude the possibility of wildfire.

Consider the following network (called a shuffle-exchange network) depicted in Figure 2. There are N (a multiple of 4) inputs numbered 0,1, ... ,N-1, and N outputs also numbered 0 to N-1, and each output connected to the corresponding input. The routing scheme is that both inputs $j =$ 0,2, ... , $N/2 - 2$ and $j + 1$ are connected to both outputs $2j$ and $2j + 2$, while inputs $j = N/2$, $N/2 + 2$, ... , $N - 2$ and $j + 1$ are connected to both $2j - N + 1$ and $2j - N + 3$. Notice that the shuffle-exchange network looks locally like a binary tree, but has only N nodes, so that the branching factor is locally 2 but is only 1 over the long run. Nonetheless, we model this case by taking $b = 2$.

We now construct our example of a wildfire cascade. We suppose that at time 0, node 0 has a rollback of size $x_0$. The rollback is passed through the shuffle-exchange network, with the following rules:

1. Each output receives the rollback size of each of its inputs.
2. The output adds a random time (see step 3.) to each positive input. Each output then takes the *larger* of the resulting numbers as the size of the rollback it has. The outputs then pass their values to inputs for the next computation cycle.
3. The independent random times are generated as i.i.d. Bernoulli random variables, with $P(W = +1) = q$ and $P(W = -K) = 1 - q$.

We see that in the case $K = 1$, $E(W) = 2q - 1$ which is negative whenever $q < 1/2$. As we mentioned, the random walk with a barrier on a binary branching tree is infinite with positive probability whenever $q \geq 0.067$; hence for N a large but finite number, we expect that wildfire cascading may exist in the present example, for some range of $q$ in $(0.067, 0.5]$.

We have simulated the aforementioned system for several values of $K$. The results are summarized in Figure 3 and compared with the theoretical approximation using the second example above. In the simulation, values of $q$ smaller then those represented by the dashed line resulted in the rollback dying out (stability), whereas values larger than those represented by the dotted line resulted in the rollback expanding to all nodes, as well as growing in size (instability).

The reader may wish to try this simulation to see for himself how a wildfire cascade may develop. The code we used can be found in Appendix A. Our random number generator is called "uni"; you may wish to use your own. With $K = 1$, the weight q set to .1 and the initial rollback size set to 10, one observes a gradual growth of the rollback, and notes that it quickly spreads to involve *all* the nodes in the network with 1024 nodes. If the weight is lowered to .07 (which is very close to the stability limit for an infinite binary tree) and the number of nodes is again 1024, then the rollback dies. However, it takes a surprising number of iterations for it to do so (on the order

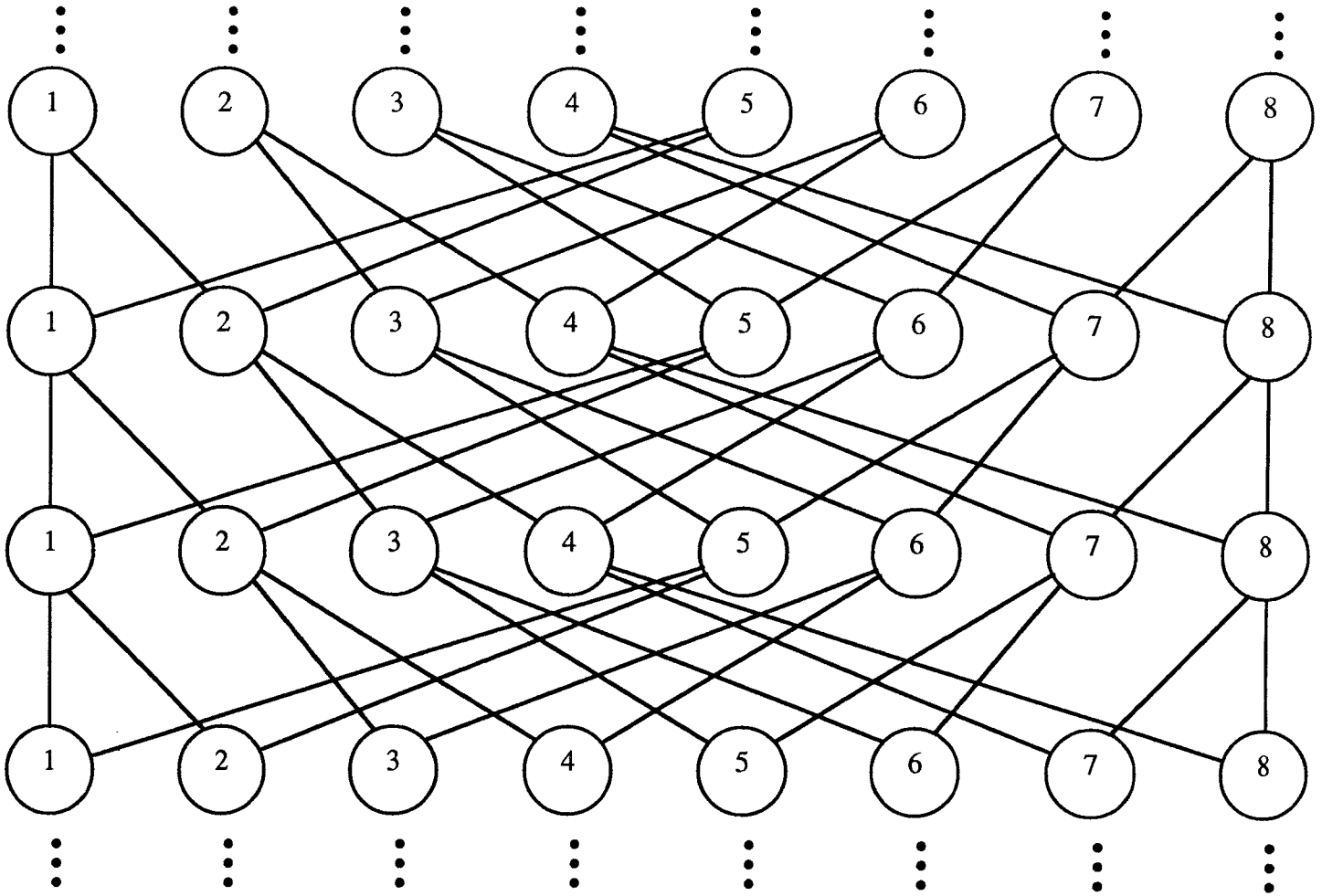


**Figure 2:** A Shuffle-Exchange Network

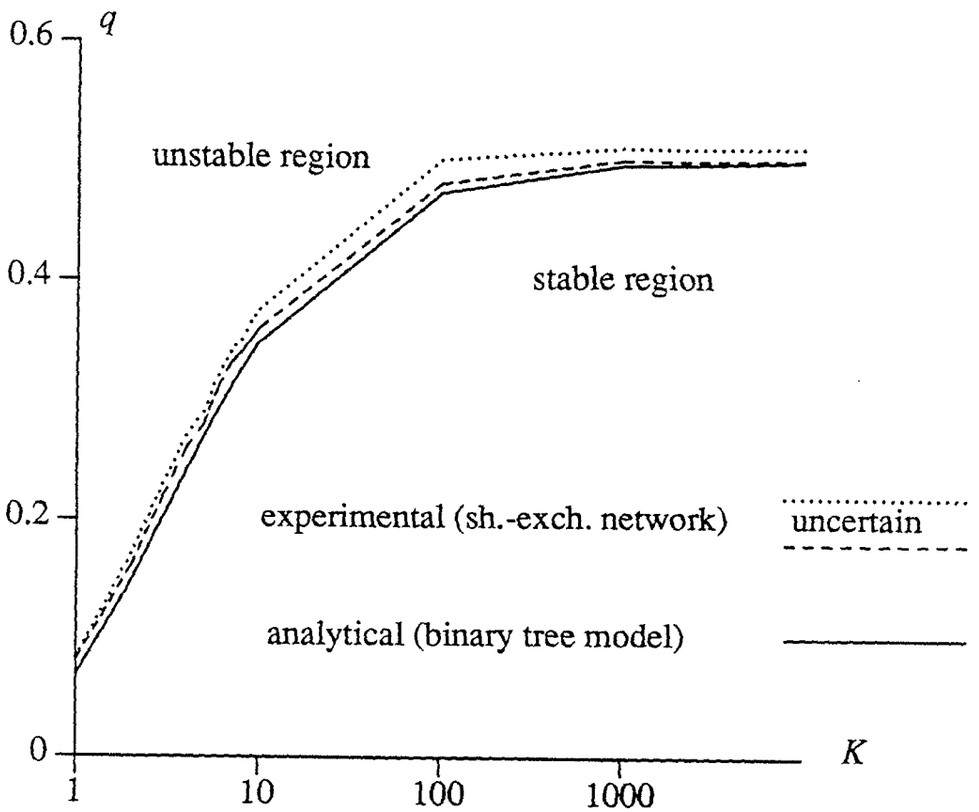


Figure 3: Shuffle Exchange vs. B.R.W.W.B.

of 1000), in view of the fact that 93 out of every 100 times, rollback decreases.

Examples of gushing cascades are easily derived from the examples above, by making $W$ large enough. We do not pursue this topic further.

### 3.5 EFFECT OF FAILURE MODES ON EFFICIENCY

In a "Moving Time Window" or "Bounded Lag" rollback scheme, such as "Filtered Rollback" presented here, there is an upper bound on the size of rollbacks that the simulation will permit. That is, there is a bound on how far out of synchronization the processors can be. In a finite system, with random rollback sizes, this may imply that all rollback trees with negative mean (such as wildfire) will eventually die. It *does not*, however, mean that every such simulation is efficient. Efficiency comes from at least two separate issues, that of how many extraneous events are generated, and how quickly time may progress in the system. How quickly does a rollback simulation progress? The first idea is to imagine the progress of the minimum time node, which presumably does not roll back. Accordingly, the minimum time in the system would seem to progress inversely to the workload of the minimum time node. In the echo example, rollback produced a high workload; specifically, the workload increased as $\sqrt{t}$. It may also cause inefficiency in another way. The marker of the "minimum time" node may migrate quickly among the nodes, and so the minimum time does not progress as fast as the local time of the minimum time node. This happened in the "zero efficiency" example. In wildfire and gushing cascading, both phenomena appear.

Rather than analyzing each particular scenario, our analysis in Section 5 directly bounds from above the workload by bounding the number of events to simulate. This estimate, coupled with an estimate of event density, gives a bound on the rate of progress.

In case there is no cascading or echoing, all the rollback trees are small, and we expect that the simulation will progress at a linear rate, with only a fixed (and not too large) fraction of processed events being erroneous. When a rollback tree becomes too large, or lives forever, then a much larger fraction of the processed events are erroneous. Theoretically we might even have an explosion in the number of erroneous events, so that the simulation bogs down and never passes a finite point in time, but in practice we expect that the simulation will simply slow down to a much slower linear rate, with a much higher fraction of erroneous events. A simulation will cross into this mode of operation suddenly. Furthermore, a pilot simulation may not show the instability when a full-scale study does. We hope that our identification of the parameters $b$ and $h$ will guide the simulationist in choosing a correct scaling.

### 3.6 SOME CONSEQUENCES OF THEOREM 2.

Theorem 2 gives bounds on the maximum rollback size in a tree. This has at least two direct applications to rollback schemes. One application is in a technical point in the proof of efficiency; see Appendix B, second paragraph. There it is used to show that even when a node rolls back, it does not have to schedule very many antievents, because the amount of time it needs to roll back is essentially bounded by the initial rollback size. This means that communication will not be a bottleneck.

The second application is an estimate of the memory requirements of a rollback algorithm. Generally, there is a floor time, also known as GVT, the smallest local time among all processors, and in the worst case a node can be rolled back as far as the floor time. Hence each node must keep in memory a list of events processed since the floor time. This is generally a vast overestimate of the amount of memory actually required. It may be that memory is kept in two different locations for rollback. A small amount is kept locally, for the more common situation of a small rollback, and the remainder is kept at a slower, large memory facility. Theorem 2 shows that the amount of memory that needs to be kept locally can be accurately estimated by the size of initial rollbacks. This may be easier to estimate or measure than the maximum size of rollbacks in a tree; for example, trees may interact, making size estimates for any one tree hard to figure.

## 4. FILTERED ROLLBACK

We now present the Filtered Rollback algorithm. Below we will show that the Branching Random Walk model is an upper bound to the behavior of the simulation system under the Filtered Rollback algorithm. We will thus be able to prove in Section 5 that Filtered Rollback can be tuned so as to be efficient in the sense of equation (2). Although our mathematical model and tools apply to other algorithms as well, we must introduce a specific algorithm, machine type and class of problems in order to show in a rigorous fashion that our Branching Random Walk model genuinely models rollback cascading. Extensions are discussed in Section 6.

### 4.1 A CONSERVATIVE ALGORITHM

The Filtered Rollback algorithm is a direct extension of the conservative Bounded Lag algorithm [Lb, Lc] described below. As in Section 1.2, we consider a system with nodes $1, 2, \ldots, N$ (e.g., a network) and assume that each node $i$ is permanently hosted by its own $\mathrm{PE_i}$. Node $i$ is represented in the simulation by its (possibly empty) set $\Pi_i$ of tentative future events, each event $e$ being a record $e = (time(e),\ contents(e))$, by $T(i)$, the minimum of the event times in $\Pi_i$; and by a bound $\alpha(i)$ of the earliest time when the history at node $i$ can be affected by other nodes. Note that the event

list $\Pi_i$ changes as nodes schedule events to be processed at $i$, and node $i$ completes processing events in $\Pi_i$. The code in Figure 4 is executed by each $PE_i$. The execution is asynchronous unless synchronization is explicitly specified and different PEs may concurrently execute different statements, e.g., $PE_1$ can execute statement 5 while $PE_2$ executes statement 6. The main principle in the algorithm is that a PE processes its events in parallel and independently of the other PEs, but only up to local time $T(i) < \alpha(i)$. This guarantees that no rollback need ever be performed.

```
initially: floor = 0, some Π_i ≠ ∅,
           time(e) ≥ 0 for each e ∈ ⋃_{1≤i≤N} Π_i,
           T(i) = min_{e_i ∈ Π_i} time(e) if Π_i ≠ ∅
           T(i) = +∞   if Π_i = ∅

1.  while floor < end_time do {
2.      compute bound α(i) of the earliest time
            when the history at node i can be
            affected by the other nodes;
3.      synchronize;
4.      while T(i) ≤ floor + B and
            T(i) < α(i) do {
5.              process events e with time T(i);
                if required, schedule new events
                for node i or delete some events
                from Π_i and/or schedule
                or delete some events e for
                other node j and insert them
                into Π_j;
6.          delete the processed events from Π_i
                and compute new T(i);
            };
7.      synchronize ;
8.      compute floor ← min_{1≤i≤N} T(i)
            and broadcast floor to all nodes ;
9.      synchronize ;
        }
```

Figure 4: The Bounded Lag Algorithm

A bound $\alpha(i)$ can be computed in several ways; the easiest to explain (not the most efficient) is the following. Let $d(i,j)$ be the *minimum propagation delay* from node $i$ to node $j$, and $S\downarrow(i,B)$ the incoming reachability sphere of node $i$ with radius $B$; i.e., the set of nodes $j$ such that $d(j,i) \leq B$. Then set

$$\alpha(i) := \min_{j \in S\downarrow(i,B)\ ,\ j \neq i} \{d(j,i) + \min\{T(j),\ d(i,j) + T(i)\}\} \tag{4}$$

If the reader will take the time to understand steps 2–9 of Bounded Lag, he should have an easier time with Filtered Rollback; the corresponding steps of both algorithms are numbered identically, and the code of Filtered Rollback can be understood as an incremental change from the code of Bounded Lag.

Bounded Lag works in synchronized cycles. At the end of the previous cycle, the $floor \leftarrow \min_{1 \leq i \leq N} T(i)$ is computed, which is the smallest overall scheduled event time. At the beginning of the next iteration, each node $i$ calculates the earliest time $\alpha(i)$ its neighbors can affect it; this comes from the propagation delay from node to node, and may change during the course of the simulation. For example, in a nonpreemptive queuing network, the remaining service time of a customer in service is a bound on how soon one node (queue) can affect another. Given $\alpha(i)$, a node is free to simulate as fast as it wishes up to the smaller of $\alpha(i)$ and $floor + B$. The node proceeds, communicating haphazardly, secure in the knowledge that nodes cannot bother each other during this safety period. Then the nodes synchronize, meaning that they all agree to stop simulating and to go to the next step, which is to compute the new $floor$, and then the new $\alpha(i)$, to start the cycle anew. It is not hard to see that a simulation performed in this fashion is correct, and that $floor$ must increase from cycle to cycle. It is not as clear that this is a good (efficient, fast) scheme. For a discussion of when Bounded Lag is efficient, see [Lc].

Observe that while the algorithm is synchronous, it may simulate an asynchronous system. Specifically, "synchronize" statements *do not* mean that all the nodes agree on what time is being simulated; rather, they agree that they are all at the "synchronize" step of the algorithm. The Filtered Rollback algorithm inherits this property.

## 4.2 FILTERED ROLLBACK

We propose that an amalgamation of Time Warp and Bounded Lag [Lb] may overcome the weaknesses of each. For discussion of these weaknesses see Section 1.3. Filtered Rollback is this combination. It has a tunable filter, which we shall prove (under appropriate assumptions) has settings which make the scheme efficient and eliminate unbounded echoing, unbounded cascading, and assure the applicability of the scheme.

Filtered Rollback is defined in Figure 5, and illustrated by the flowchart in Figure 6; as mentioned in Section 4.1, the numbering scheme is chosen to correspond to the Bounded Lag algorithm. Armed with an understanding of the Bounded Lag algorithm, we now provide some remarks on the algorithm, as presented in Figure 5.

Each node in the simulator is supposed to be concurrently implementing the algorithm as

```
initially: floor = 0, some Π_i ≠ ∅,
           Π'_i = ∅ and t(i) = 0 for i = 1,...N,
           time(e) ≥ 0 for each e ∈ ⋃_{1≤i≤N} Π_i,
           T(i) = min_{e_i∈Π_i} time(e) if Π_i ≠ ∅
           T(i) = +∞   if Π_i = ∅
1.   while floor < end_time do {
2.          compute approximation α(i) of the earliest time
                when the history at node i can be
                affected by the other nodes ;
3.          synchronize;
3a.         if t(i) > T(i) then {
3b.             recover the state of node i including Π_i the
                    node had just before simulated time T(i) ;
3c.             recover all events with simulated time at
                    and after time T(i) previously incorporated
                    in Π_i in statement 7a by previous Π'_i
                    and insert the recovered events
                    into current Π'_i ;
3d.             for each event e scheduled previously
                    by node i in a different Π_j on
                    the interval of simulated time [T(i), t(i)],
                    create the matching antievent ē and
                    insert it into Π'_j;
                t(i) ← T(i);
            } else
4.          while T(i) ≤ floor + B and
                T(i) < α(i) do {
5.                  process events e with time T(i);
                  if required, schedule new events for node i
                      or delete some events from Π_i and/or
                      schedule some events or antievents e for
                      other node j and insert them into Π'_j;
5a.               t(i) ← T(i);
6.                delete the processed events from Π_i
                      and compute new T(i);
                };
7.          synchronize ;
7a.         Π_i ← Π_i ⋃ Π'_i; Π'_i ← ∅;
                compute new T(i);
8.          compute floor ← min_{1≤i≤N} T(i)
                and broadcast floor to all nodes ;
9.          synchronize ;
            }
```

Figure 5: Filtered Rollback

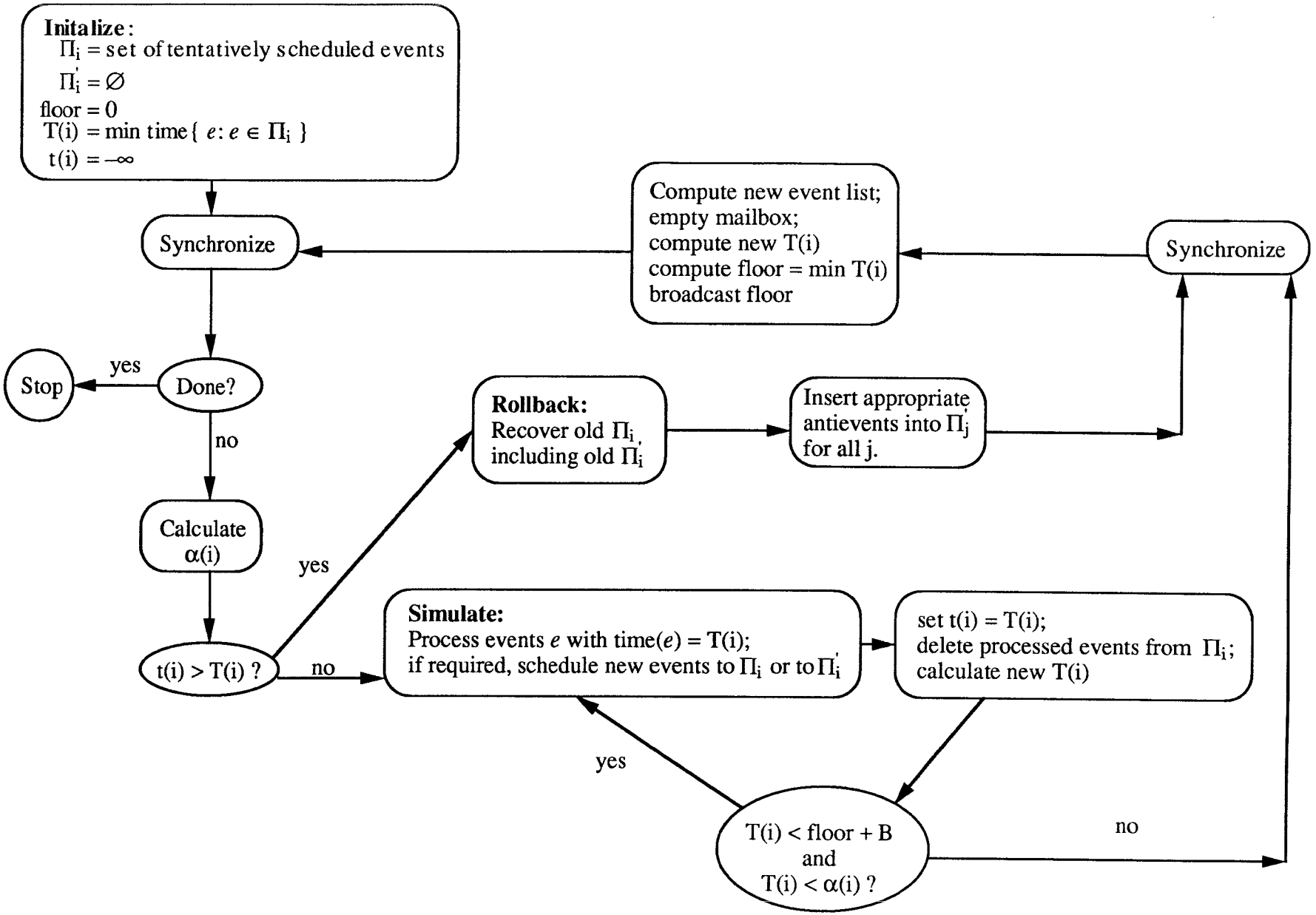


**Figure 6:** Flowchart of Filtered Rollback

described. The $\Pi_i$ and $\Pi'_i$ are events lists that use local memory. Their events $e$ have to be maintained until *floor* $>$ *time*$(e)$, to facilitate rollback. Suppose that the estimate $\alpha(i)$ ($\alpha(i)$ is the eponymous filter) is computed in step 2 by an efficient scheme. Such a scheme will be described in the next paragraph. In contrast with the Bounded Lag algorithm, $\alpha(i)$ is an approximation, so that rollbacks may still occur; $\alpha(i)$ is only used to control the *frequency of occurrence* of rollbacks. After forming their estimates, the nodes synchronize; that is, agree to perform step 3. The variable $t(i)$ represents the time of the last event simulated at node $i$, while $T(i)$ is the time of the next event scheduled to be processed. Hence $t(i) > T(i)$ implies that node $i$ needs to roll back. Each node now embarks on the job of either *simulating* some new events in steps 4-6, *waiting* at step 7 if the local clock is higher than either the global lag bound or the local estimate $\alpha(i)$, or *rolling back* if need be in steps 3a-3c. Then the nodes synchronize, decide if they are done or not, and either continue the loop or terminate. The estimate $\alpha(i)$ is used to keep a processor from plowing ahead when there is a good chance that anything it computes will have to be undone by a rollback. If the estimate is 100% accurate, then the algorithm actually never rolls back, and is equivalent to Bounded Lag. Otherwise, whenever the estimate $\alpha(i)$ is violated, node $i$ may need to roll back, and may induce rollback in other nodes. The accuracy of the estimate $\alpha(i)$ is a tunable part of the algorithm, and what we prove later is that under appropriate statistical assumptions on the behavior of the simulation, the estimate may be adjusted so that the algorithm as a whole is efficient.

One way to generate $\alpha(i)$ used in steps 2 and 4 is the following. Suppose an event $e$ at node $i$ may generate new events at node $j$ at time $time(e) + t_e(i,j)$. Let us also suppose that each such event $e$ has a corresponding estimate $d_e(i,j)$ of the propagation delay. Again, this is an estimate, not a bound. We also suppose that there is a computable estimate $d(i,j) = \inf_e d_e(i,j)$. Then we define $\alpha(i)$ through (4). The delays $d(i,j)$ are supposed to satisfy the triangle inequality: $d(i,j) + d(j,k) \geq d(i,k)$ for any three nodes $i$, $j$, and $k$.

It may be shown that *floor* in Filtered Rollback increases from cycle to cycle, assuming that a rollback only goes as far back as the earliest erroneous event. The floor gives the minimum time of events that need to be stored in memory in case of rollback. The notion of a floor is useful in analyzing both the correctness of the algorithm, and its memory usage, but we do not pursue these issues here.

## 5. PROOF OF EFFICIENCY

### 5.1 PREFATORY NOTIONS

This section contains a proof of efficiency of the Filtered Rollback algorithm under appropriate

assumptions. The assumptions are detailed later; for now, we will give an overview of the ideas, strengths, and weaknesses of our analysis.

Our point of view is that there are $O(N)$ events to simulate, with $N$ nodes where the events take place*. Rollback causes us to simulate some events more than once, and possibly to simulate some fallacious events. Our main contribution is to identify reasonable stochastic assumptions on the underlying system such that the total number of simulated events, correct, false or repeated, is only $O(N)$. The next subsection details this argument by carefully classifying and counting the simulated events.

The first set of assumptions, **1–4**, involves the structure of the simulated system and the simulating machine. Roughly speaking, condition **5** is that no processor becomes a bottleneck. Condition **6** is that there is enough concurrency in the system to keep $O(N)$ simulating processors busy all the time. Although these conditions seem reasonable, they assume something about the evolution of the simulation. We would have preferred to derive these properties strictly from assumptions on the simulated system and the definition of Filtered Rollback, but we are as yet unable to do so.

Underlying all of the assumptions for our efficiency proof is the notion that as $N$, the number of nodes, tends to infinity, the local behavior of the system does not change. For example, assumption **1** is that the amount of work involved in processing an event does not grow with $N$. Assumptions **2** through **6** are stated twice, as $\mathbf{n}'$ and as $\mathbf{n}$. The $\mathbf{n}'$ statements are simpler and more intuitive, but are more restrictive than the $\mathbf{n}$ statements, and are mainly included for the reader who wishes to avoid involved probabilistic reasoning. The actual proofs are based on the weaker $\mathbf{n}$ statements.

Here is the notation used in our proof of efficiency. $I$ stands for the set of events to simulate (see 5.2), and $|S|$ is the number of elements of the set $S$. A *cycle* of the Filtered Rollback algorithm goes from Step 9, say, through Steps 2–8 and arriving back at Step 9. Finally, for each event $e$, we denote by $t_e$ the time when the event occurs. $t(i,j)$ is the earliest time an event at $i$ can affect node $j$. Although this depends on the specific event, we use $t(i,j)$ as a "generic" variable.

Here is the list of assumptions. **A discussion of each follows.**

**Assumptions : $N = \#$ nodes**

There exist positive constants $\beta, C, \eta, \delta, \sigma, D, \rho, \epsilon, a_0, n_0, r$ all independent of $N$, node index, and iteration number, and estimates $d(i,j) > 0$ such that

* We eliminate this restriction in Section 5.5, but use it in this section in order to simplify the discussion and the proof.

**1'** (amount of work to process an event) $\leq a_0$ and scheduling an event takes a unit of work.

**1** (amount of work to process an event $e$) $\leq U(e)$, where the $U(e)$ are i.i.d. and satisfy $P(U(e) \geq a) \leq Ce^{-\sigma a}$ and scheduling an event takes a unit of work.

**2'** $t(i,j) \geq \eta$ for $i \neq j$.

**2** $\lim_{B \downarrow 0} P$ (an event $e$ seeds a rollback tree) $= 0$ uniformly in $N$ (recall $B$ is the lag bound).

**3'** $d(i,j) - t(i,j) = W_{ij}$ are i.i.d., and $W_{ij}$ satisfy A1), A2) and A3) given in Section 2.1.

**3** $d(i,j) - t(i,j) \leq W_{ij}$ a.s., where $W_{ij}$ are i.i.d., and satisfy A1), A2) and A3).

**4'** $D < e^h$ where $h$ is defined in (3), and $D < \infty$ bounds the number of nodes connected to a given node.

**4** There exists i.i.d. random variables $\{y_{n,j}\}$ with $E\left(y_{1,1}^{1+\beta}\right) < \infty$, $b = E(y_{1,1}) < e^h$ and $P$ (node $i$ schedules at least $l$ events at other nodes at an iteration) $\leq P(y_{1,1} \geq l)$.

**5'** The number of events within $\Pi_i$ subject to processing at each iteration does not exceed $\delta$.

**5** Let $\Pi_i(s, \Delta)$ denote the set of events (including those which will be cancelled by rollbacks) which are scheduled between times $s$ and $s + \Delta$. For any $\Delta > 0$,

$$P(|\Pi_i(s,\Delta)| \geq n_0 + t \text{ for some } s) \leq e^{-\frac{rt}{\Delta}} .$$

**6'** At least $\epsilon N$ nodes are busy at each cycle.

**6** $P$ (number of nodes busy at each cycle $\geq \epsilon N$) $\geq \rho$.

Assumption **1** upper bounds the work to process an event by an independent random variable with an exponential tail. This means, in particular, that each event schedules at most an exponentially bounded number of other events. Together with Assumption **5**, this means that each node only communicates with an exponentially bounded number of other nodes at each iteration of the Filtered Rollback algorithm. This will be used to bound the execution time of a cycle of the algorithm. It is also much more general than most simulations need: usually, each node only communicates directly with a fixed finite set of other nodes.

Assumption **2** is that, as the processors' local times become closely coupled, the probability of starting a rollback tend uniformly to zero. It is weaker than having a bounded speed of propagation within the system (which is assumption **2'**). It means as $B \downarrow 0$, the simulation becomes more serial, and so all rollbacks disappear.

Assumption **3** upper bounds the errors in estimating the propagation delays between the nodes by an i.i.d. sequence. The delays were introduced and discussed in Sections 4.1 and 4.2. We believe that when it is not possible to provide estimates $d(i,j)$ satisfying **3**, then the system is really not

very parallel. The reason is that **3** is violated when events at different nodes are highly correlated and tend to happen almost simultaneously. This implies either that the simulated system is serial, or that our partitioning into parallel nodes should be redone so as to make such correlated events occur only locally within single nodes. Also, the assumption that the $W_{ij}$ are independent is a replacement for an ergodicity assumption. For example, suppose that $D = 1$ (see Assumption **4'**), $t(i,j) = 100$ with probability .99, and $t(i,j) = 1$ with probability .01. Then whether or not the estimate $d(i,j) = E(t(i,j)) - 1.01 = 98$ works depends on how well mixed the values of $t(i,j)$ are. Assumption **3** also applies to the causality propagation of noninherent events. Assumption **3'** is stronger than assumption **3** in that the errors in the estimation of the propagation delays $t(i,j)$ are assumed strictly i.i.d., and not just bounded by an i.i.d. sequence.

Assumption **4** stochastically bounds the number of nodes a given node communicates with during an iteration of the algorithm. The bounding distribution has slightly more than one moment. In fact, the bound would be a constant in most cases. The much cruder Assumption **4'** simply bounds the degree.

Both **5** and **6** are assumptions on the density of events. **5** upper bounds the density in time, while **6** lower bounds the density in space (across nodes). According to **5**, events don't pile up on a node more than a bulk Poisson process would. The cruder **5'** requires each node to have only a bounded number of events to process at each cycle. Events are well spread out over the nodes, according to **6** or **6'**. This spread holds uniformly over the cycles under **6'**, and on average under **6**. The reason that **6** doesn't necessarily imply that the simulation is efficient is that, as in the echo example, most of the work may be performed on fallacious events. Note that both sets of assumptions apply to both inherent and noninherent events. We are not very satisfied with the latter point, because we feel that it might be derivable from the definition of the algorithm.

### 5.2 NUMBER OF PROCESSED EVENTS

In analyzing the efficiency of a rollback-based algorithm, one needs to count the number of events that are processed. Note that a rollback algorithm may sweep over the same simulated time more than once. Hence the same event can be processed several times. Moreover, during these sweeps some erroneous events may be introduced and processed. We need to distinguish between *inherent events*, those which actually occur and are never rolled back, and *processed events*, which are all events, inherent or not, each counted as many times as processed. Processed events that are not inherent are called noninherent*. Let us denote by $I$ the set of inherent events and by $\Pi$ the set of

* "inherent" vs. "noninherent" events are sometimes called "speculative" vs. "committed"

processed events. (The goal of our simulation is eventually to produce only the inherent events.) Then $I \subset \Pi$ and all events in $\Pi \setminus I$ are cancelled by rollbacks.

We need to describe carefully the connection between events, nodes, and rollback trees. Suppose node $A$ processes an event $e_A$ which schedules an event $e_Z$ at node $Z$. Suppose further that $t_{e_Z} < t(Z)$ (for the definition see step 5a of the Filtered Rollback algorithm), so node $Z$ has to roll back. We say that event $e_A$ *seeds* the rollback tree, even though it is not necessarily a direct part of any rollback (i.e. it is not an antievent, or necessarily cancelled by an antievent).

We now give a finer classification of the set $\Pi$. We decompose $\Pi$ by classifying the degree to which each event is "non-inherent". We do this by constructing sets $I^{(k)}$ such that

$$\Pi = I^{(0)} \cup I^{(1)} \cup I^{(2)} \cup \ldots, \quad \text{and} \quad I^{(i)} \cap I^{(j)} = \emptyset \text{ when } i \neq j \ .$$

Formally, define $I^{(0)} \stackrel{def}{=} I$. Recursively define $I^{(k+1)}$ as the set of events which are part of trees (i.e. are cancelled by rollback trees) seeded by events in $I^{(k)}, k = 0, 1, \ldots$ Thus all events not in $I^{(0)}$ are noninherent events. In the example above, if $e_A$ is an inherent event, then the set of events $S_A$ cancelled by the rollback seeded by $e_A$ are in $I^{(1)}$. However, if $e_A$ is not inherent, and is later cancelled by a rollback seeded by, say, an event in $I^{(0)}$, then we classify $e_A$ as belonging to $I^{(1)}$ and the events in $S_A$ as belonging to $I^{(2)}$.

We bound the size of $|\Pi|$ by producing estimates of the size of $|I^{(k)}|$. Our analysis is based on assumption **2** that each event has the same probability of rooting a rollback tree, (or is bounded by the same probability) and assumption **3** that all rollback trees in the various $I^{(k)}$ are statistically identical (or at least bounded by an identical distribution). This assumption seems fairly conservative to us. Events in $I^{(k)}$ with $k$ large are quite out of synchronization with the rest of the simulation, and so may be expected to start *fewer* and *smaller* rollback trees than events in $I^{(k)}$ with small $k$.

**Theorem 3.** *Suppose that the total number of events to simulate is $O(N)$. Then under assumptions* **2, 3, 4** *the expected total number of events processed during the course of the simulation is $O(N)$.*

**Proof.** From (9) and the proof of Theorem 4 of Appendix B, there is a set of estimates $\{d(i,j)\}$ and a $\beta < 1$ so that

$$E|I^{(k+1)}| \leq \beta \, E|I^{(k)}| \tag{5}$$

events, or "correct" vs. "incorrect" events.

Hence

$$E|\Pi| = E\sum_{k=0}^{\infty} |I^{(k)}| \leq |I^{(0)}| \sum_{k=0}^{\infty} \beta^k \leq \frac{C}{1-\beta} N = O(N),$$

the second inequality coming from the assumption that the total number of events to simulate is $O(N)$. ■

The idea behind Theorem 3 is that as simulation becomes more conservative, the number of rollback trees decreases, and the size of rollback trees does not increase. This seems to indicate that we should do purely conservative simulation, but this is not so. First, recall from Section 1.3 that some systems are impossible to simulate without any rollback at all. Another reason to have rollback is to gain efficiency. A perfectly conservative scheme is to have only the event with smallest local time be processed; this idles all but one processor in general. On the other hand, as we know from the analysis of wildfire cascading (see Table 3 — the Shuffle-Exchange example) schemes which are too optimistic may be inefficient due to large overheads, and may even fail. We expect that a modestly optimistic scheme will perform better than either, by having a few rollbacks in exchange for more parallelism.

### 5.3 ASSUMPTIONS ABOUT THE MULTIPROCESSOR

Here is our model of computation. We assume that a size $N$ multiprocessor can:

1) Synchronize in (deterministic) time $O(\log N)$.
2) Compute and broadcast $\min_{1 \leq i \leq N} T(i)$ in time $O(\log N)$; here processor $i$ has variable $T(i)$.
3) Support concurrent pair-wise communication of messages with size up to $K$ each, in time $O(\log N + K)$. (Pair-wise means each processor sends to at most one other processor, receives from at most one other, and does not simultaneously send and receive.)
4) Have all processors working independently (no communication) at full speed; rate one, say, so that the time to process a size one event takes one unit of time, independently of $N$.

Many standard models of multiprocessors satisfy these assumptions; see, e.g. [KRS]. In Section 5.4 we examine a standard PRAM model, but for now we use the listed assumptions as our machine model.

### 5.4 PROOF OF EFFICIENCY OF FILTERED ROLLBACK

Our proof of the efficiency of Filtered Rollback involves two basic elements: counting events, and estimating the time to process events. Theorems 1,2, and 3 show that, appropriately tuned, Filtered Rollback processes only $O(N)$ events when there are $O(N)$ inherent events. Assumption **6**

states that $O(N)$ events are processed per cycle; hence only $O(1)$ cycles are needed to finish the computation. We show in this section that the mean cycle time is $O(\log N)$. This gives a total time to complete the algorithm of $O(\log N)$. We relax the assumption that there are $O(N)$ inherent events in the next section.

**Theorem 4.** *Suppose that the total number of events to simulate is $O(N)$. Under assumptions* **1-6**, *a size $N$ multiprocessor will execute Filtered Rollback in mean time $\tau$ satisfying*

$$\tau = O(\log N).$$

The proof of Theorem 4 is given in Appendix B. We now give the main ideas of the proof. Let $Y$ be the number of cycles the algorithm uses to execute the simulation, and let $U_i$ be the time to execute cycle $i$. We have

$$\tau = E \sum_{i=1}^{Y} U_i.$$

If $Y$ and $U_i$ were not random, were independent, or more generally if Wald's Lemma applied to this, we would have

$$\tau = E(Y)\, E(U_i).$$

Then, as noted in the first paragraph of this subsection, $E(Y) = O(1)$, and a close examination of the steps in Filtered Rollback given below shows that $E(U_i) = O(\log N)$.

Clearly $E(U_i)$ is at least $O(\log N)$ since each "synchronize" takes $O(\log N)$ time; we simply have to be careful that the rollback cycle (3a–3c in the algorithm) and the simulation cycle (4–6) do not take more than $O(\log N)$. The idea is to use the exponential bounds on the tails of all the random events in the problem. It is well known that for i.i.d. exponentially distributed random variables $Z_i$,

$$E(\max_{1 \le i \le N} Z_i) = O(\log N)$$
$$Var(\max_{1 \le i \le N} Z_i) = O(1). \tag{6}$$

Now the time to execute each step in Filtered Rollback at any node is either assumed to have an exponentially small tail, or is proved in Appendix B to have one. Equation (6) shows that each step, including busy waits, takes on average no more than $O(\log N)$ time, and has small variability as shown by the variance estimate in (6). The proof is very simple in conception: each step of Filtered Rollback takes polylog time per event, and we assume (in **5** and **6**) that each node only has a few events to process, but that many nodes are busy processing.

The proof is not complete because Wald's lemma does not apply to our system. This necessitates assumption **4** on the existence of a $1+\beta$ moment for $y_{n,j}$ as well as the rather involved proof of Appendix B. This completes our discussion of the proof of equation (F), and we complete the technical arguments for Theorem 4 in Appendix B. The arguments are fairly similar to those in the efficiency proof of Bounded Lag in [Lc], except the existence of rollback makes the delicate stochastic analysis necessary.

### 5.5 FILTERED ROLLBACK ON A PRAM

Our efficiency estimates for Filtered Rollback can be extended, or specialized, to the familiar setting of an algorithm executing on a PRAM. A PRAM is a standard model of a shared memory multiprocessor. We will show that for the weakest model of PRAM, namely exclusive read, exclusive write (EREW), the mean time $\tau$ to simulate a system with $N$ nodes and $R$ inherent events on $p$ processors with $R \geq N \geq p$ satisfies

$$\tau = O\left(\frac{R}{p} + \frac{R}{N}\log\ p\right). \tag{7}$$

The idea of the proof is similar to Brent's Theorem. By examining the steps of the proof, the reader may find that (7) holds for Filtered Rollback executing on different models of a multiprocessor. A comprehensive discussion of these issues may be found in [KRS]. Note that an EREW PRAM satisfies our multiprocessor model (Section 6.3). The point of this Section is that $p$ may be much smaller than $N$, and $R$ may not scale linearly with $N$. Equation (7) establishes Filtered Rollback as an ENC algorithm in the hierarchy established in [KRS], where complexity is measured in terms of $N$; this is the highest category in terms of both speedup and efficiency. We begin by breaking up the simulation into $\frac{R}{N}$ sub-simulations. Each of these satisfies Assumption 1 of Section 6.1, so on $N$ processors would take $O(\log N)$ time. Thus if $p = N$ we have established (7). We prove (7) by having a $p(< N)$ processor PRAM simulate an $N$ processor machine. We assign $\frac{N}{p}$ nodes to each processor. The simulation runs by having each of the $p$ processors perform a Filtered Rollback cycle for each of the $\frac{N}{p}$ nodes it hosts.

The only difference between the simulation and an $N$-processor implementation is the communication between nodes. For nodes hosted by one processor it is easy to arrange that there is no communication delay, synchronization penalty, or time lost due to non-concurrency (some simulated processors waiting for others to finish). We simply have to show that the communication between simulated processors hosted on different actual processors does not become a bottleneck. Here we use assumptions **1** and **5** . We establish a separate "mailbox" (memory location) for each

communicating node pair $(i \to j)$. During the "inner loop", each processor will write $\sum_{i \le \frac{N}{p}} z_i$ messages, spread over at most $DN/p$ mailboxes, where $z_i$ is a random variable with an exponential tail. There are no concurrency problems since the mailboxes are all distinct, so this takes $T = \max_{k \le p} \sum_{i \le \frac{N}{p} D} z_{i,k}$ time. $T$ satisfies

$$E(T) \le C \left( D\frac{N}{p} + \log p \right)$$

and $T$ has an exponential tail. Synchronizing takes $O(\log p)$ time, and computing $\alpha(i)$ takes no more than $O(\frac{N}{p} + \log p)$ time. The rest of the cycles are estimated as before. Hence we may use Appendix B to establish equation (7).

## 6 CONCLUDING REMARKS

The moral of this paper is that it is possible to analyze rollback under structured conditions, and draw useful conclusions. The main one is that an intelligent combination of conservative and rollback-based schemes may be more powerful than either type in its pure form.

We conclude with a short list of unanswered questions and open topics.

1. Extend the analysis to other topologies (lattice, hypercube) which are not expander-like graphs.
2. Improve the bounds by taking into account dependencies of the change in rollback size on the local size, on the number of induced rollbacks etc.
3. Include the effect of finite system/finite memory. This and 2. involve extending the mathematical analysis of [BLSW].
4. Analyze the strategies for speeding antievents, such as [Fb].
5. Obtain a good wildfire simulation.
6. Derive assumptions **5** and/or **6**.
7. Allow for some nonuniformity and inhomogeneity in the simulated system and/or simulation.
8. Asynchronous algorithm analysis.
9. Analyze memory requirements and handling more carefully.

## ACKNOWLEDGMENTS

We thank Clyde Kruskal for suggesting the results of Section 5.5 and for outlining the arguments used there. We thank the six (!) reviewers for many constructive comments, leading to improvements in the exposition of this paper.

## REFERENCES


[AW] Alder, B.J., Wainwright, T.E., "Studies in molecular dynamics I. General method," *J. Chem. Phys.* Vol. 31 No. 2 pp. 459–466, 1959.

[A] Asmussen, S., "Some Martingale methods in the limit theory of supercritical Branching Processes", in *Branching Processes*, Edited by A. Joffe and P. Ney, Marcel Dekker, New York, pp. 1–26, 1978.

[AH] Asmussen, S. and Hering, H., *Branching Processes*, Birkhauser, 1983.

[AR] Azencott, R. and Ruget G., "Mélanges d'équations différentielles et grands écartes à la loi des grands nombres," *Z. Wahr.* **38**, pp. 1-54, 1977.

[B] Biggins, J.D., "Chernoff's theorem in the branching random walk", *J. Appl. Prob.* **14**, pp. 630-636, 1977.

[BLSW] Biggins, J.D., Lubachevsky, B.D., Shwartz, A. and Weiss A., "A Branching Random Walk with a Barrier", *Annals of Appl. Prob.*, to appear, 1991.

[Ch] Chernoff, H., "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations", *Ann. Math. Statist.*, **23**, 494–507, 1952.

[Fa] Fujimoto, R.M., "Performance measurements of distributed simulation strategies," *Proceedings of the 1988 SCS Multiconference*, Simulation Series, SCS **19** No. 3 pp. 14–20.

[Fb] Fujimoto, R.M., "Time warp on a shared memory multiprocessor," in *Proceedings of 1989 Int. Conf. Parallel Processing*, pp. III-242–III-249, 1989.

[Fc] Fujimoto, R.M., "Parallel discrete event simulation," in *Proceedings of the 1989 Winter Simulation Conference*, pp. 19–28, 1989.

[Fd] Fujimoto, R.M., /lq/lq Parallel discrete event driven simulation," *Comm. ACM* **33** No. 10 pp. 31–53 (1990).

[GBJ] Gafni, A., Berry, O., Jefferson, D., "Optimized virtual synchronization", *Proc. 2nd Int. Workshop on Applied Math. and Performance/Reliability Models,* Univ. of Rome II, 229–244, 1987.

[GLM] Greenberg, A.G., Lubachevsky, B.D., Mitrani, I., "Unboundedly parallel simulations via recurrence relations," *Proc. 1990 Sigmetrics Conf.* 1990.

[HB] Hontales, P., Beckman, B. et. al., "Performance of the colliding pucks simulation of the Time Warp operating systems," *Proc. 1989 SCS Multiconference*, Simulation Series, SCS, Vol. 21 No. 2 pp. 3–7 1989.

[J] Jefferson, D.R., "Virtual time", *ACM Transactions on Programming Languages and Systems,* **7**, 3, 404–425, 1985.

[KRS] Kruskal, C.P., Rudolph, L. and Snir, M., "A complexity theory of efficient parallel algorithms,"

To appear, *Theoretical Computer Sci.*, 1990.

[LMS] Lavenberg, S., Muntz, B., Samadi, B., "Performance analysis of a rollback method for distributed simulation," *Performance 83*, 1983.

[LL] Lin, Y-B., Lazowska, E.D., "Optimality considerations for "time warp" parallel simulation," in *Distributed Simulation 1990*, pp. 29–34, SCS Simulation Series, 1990.

[Lb] Lubachevsky, B.D., "Efficient distributed event driven simulations of multiple-loop networks," *Communications of the ACM*, **32**, 1, pp.111–131, 1989.

[Lc] Lubachevsky, B.D., "Scalability of the Bounded Lag distributed discrete event simulation," in *Distributed Simulation*, B.Unger, and R.Fujimoto (eds.), SCS, *Simulation Series*, **21**, 2, 100–107, 1989.

[Ld] Lubachevsky, B.D., "Simulating colliding rigid disks in parallel using bounded lag without Time Warp," *Proc. 1990 SCS Multiconference*, Simulation Series, SCS, **22** 1 pp. 194–202, 1990.

[LSWb] Lubachevsky, B.D., Shwartz, A. and Weiss A., "The stability of a Branching Random Walk with a Barrier", EE. PUB. 748, Technion, Israel, 1990.

[LSWa] Lubachevsky, B.D., Shwartz, A. and Weiss A., "Rollback Sometimes Works ... If Filtered", in *Proceedings 1989 Winter Simulation* pp. 630–639.

[MWM] Madisetti, V., Walrand, J., Messerschmitt, D., "Synchronization in message-passing computers, models, algorithms and analysis," Simulation Series, SCS, **22**, 2, 35–48, 1990.

[MM] Mitra, D. and Mitrani, I., "Analysis and optimal performance of two message-passing parallel processors synchronized by rollback," *Performance 84*, pp. 35–50, 1984.

[N] Nicol, D.M. "Parallel discrete-event simulation of FCFS stochastic queueing networks," *Proceedings ACM SIGPLAN Symp. on Parallel Programming* pp. 124-137, 1988.

[Sa] Samadi, B., "Distributed Simulation, Algorithms and Performance Analysis" Ph.D. Dissertation, UCLA Comp. Sci. Dept., March 1985.

[Sh] Shepp, L.A., "Connectedness of certain random graphs," *Israel Journal of Mathematics* Vol. 67 pp. 23–33 1989.

[SBW] Sokol, L.M., Briscoe, D.P, and Wieland, A.P., "MTW: a strategy for scheduling discrete simulation events for concurrent execution," in *Distributed Simulation*, B.Unger, and D.Jefferson (eds.), Simulation Series, SCS, **19**, 3, 34–42, 1988.

[SS] Sokol, L.M., Stucky, B.K., "MTW: Experimental results for a constrained optimistic scheduling paradigm," in *Distributed Simulation*, D. Nicol (Ed.), Simulation Series Vol. 22, 1/2 pp. 169–173, 1990.

## APPENDIX A: A simulation program of Rollback in a Shuffle-Exchange network

Here is the code we used for simulating the Shuffle-Exchange Network (using Fortran77).

```
      dimension ix(1024), iy(1024), iw(4)
      print *,"How many nodes (make it even)?"
      read *, n
      print *,"What's the init rollback size at node 0?"
      read *,init
      print *,"How many iterations?"
      read *, niter
      print *,"What weight?"
      read *,q
      print *,"How often should I checkpoint?"
      read *,icheck
      do 10 j=2,n
   10 ix(j)=0
      ix(1)=init
      jcheck=icheck
      do 20 iter=1,niter
      do 15 jj=1,n/2
      j=jj-1
      do 12 k=1,4
      if (uni(0).lt.q) then
      iw(k)=1
      else
      iw(k)=-1
      endif
   12 continue
      if (ix(2*j+1).le.0) then
      iw(1)=0
      iw(3)=0
      endif
      if (ix(2*j+2).le.0) then
      iw(2)=0
      iw(4)=0
      endif
      iy(2*j+1)=max(ix(2*j+1)+iw(1),ix(2*j+2)+iw(2))
      iy(2*j+1)=max(iy(2*j+1),0)
      iy(2*j+2)=max(ix(2*j+1)+iw(3),ix(2*j+2)+iw(4))
      iy(2*j+2)=max(iy(2*j+2),0)
   15 continue
      do 17 j=0, n/2 -1
      ix(2*j+1)=iy(j+1)
   17 ix(2*j+2)=iy(j+1+n/2)
      if (iter.eq.jcheck) then
      print *,"n Iteration",iter
      do 19 j=0,n/2-1
      print *,2*j+1,ix(2*j+1)
   19 print *,2*j+2,ix(2*j+2)
      jcheck=jcheck+icheck
      endif
   20 continue

    end
```

## APPENDIX B: PROOF OF EFFICIENCY

We begin our proof of efficiency with a detailed look at the cycle times of Filtered Rollback. As indicated in Section 5.4, each step in the cycle takes at most $\log N$ time, and has an exponential tail. We now justify this claim. We do this by bounding separately the amount of time each step in the algorithm takes at each node. More precisely, we show, for each processor and each step, there are constants $C_0, C_1, C_2$ and $t_0$ with

$$P\ (\text{time to finish step } \geq t + t_0 + C_0 \log N) \leq C_1 e^{-C_2 t}. \tag{8}$$

We examine each step of Filtered Rollback in a separate paragraph.

Step 2 involves each node computing the approximations $\alpha(i)$ and communicating with its reachability set. Bt assumption **4** the reachability set has a polynomial tail, and therefore, since communication time is the logarithm of the size of the set (by property 2 of the multiprocessor model), the communication time has an exponential tail. The rest of the computation is $O(1)$ time by property 4 and assumption **1**. Thus the average time for step 2 is $O(\log\ N)$.

Step 3 is synchronization. This takes $O(\log\ N)$ by property 1.

Steps 3a, 3b, and 3c are rollback. A node rolling back must do two things:

b) Recover memory

c) Schedule antievents.

Step b) involves merging a previous event list (chosen depending on how far the node is rolling back) $\Pi_i$ with the union of lists $\Pi_i'$ received from that point on. Each of these lists is sorted, so the time to merge is $O(|\Pi_i| + |\Pi_i'|)$. The size of the lists has an exponentially bounded tail, by at least two different reasons: First the lists only have to be merged up to time $floor + B$, and there are an exponentially bounded number of events there by Assumption **5**. Second, the amount of time which needs correction is bounded by $\max_{n,i}\ x(n,i)$, where $x(n,i)$ is the size of a (bounding) branching random walk. Theorem 2 shows that this has an exponentially bounded tail (since $x(0,1) \leq B$), and then again use Assumption **5**. [We include this second reason because we feel that it generalizes more easily to other algorithms]. Now Step c) involves sending messages for each of the events in a time period bounded again by $\max_{n,i}\ x(n,i)$. Hence we use Assumption **5** and property 3 of the multiprocessor model to find that the time to execute Steps 3a–3c is $O(\log\ N + Z)$, where $Z$ is a random variable with an exponential tail.

Steps 4, 5, 5a, and 6 are the heart of the simulation. By Assumption **5** a node has at most $n_0 + Z_i$ events to process, which, by property 4) takes up to a unit of time each, and where $Z_i$ has

an exponential tail. Each processed event can schedule at most an exponentially bounded number of other events. Hence the time to cycle through these steps is

$$O(\log N + \text{constant} + Z_i) = O(\log N + Z_i).$$

Step 7 is a synchronize. Again, this takes $O(\log N)$ time.

Step 7a updates the event list. First we sort the list $\Pi'$. There are up to $Bn_0 + \sum_{j \in S\downarrow} Z_i$ events to sort, $|S\downarrow| \leq B$, so this sort can be accomplished in $(Bn_0 + \sum_{S\downarrow} Z_j) \log (Bn_0 + \sum_{S\downarrow} Z_j)$ time; this clearly has an exponential tail. Merging two sorted lists takes linear time (see Steps 3a–3c), and so the time to perform Step 7a has an exponential tail.

Step 8 is computing and broadcasting the new *floor*. This takes $O(\log N)$ time by Assumption 2 of the multiprocessor model.

Step 9 is the final synchronize. Again this takes $O(\log N)$ time. This completes the proof of equation (8).

Let $W$ be the total number of events generated in the simulation. Then as argued in Section 5.2, $W = \sum_{k \geq 0} \mid I^{(k)} \mid$. We construct a Branching Process which bounds $\mid I^{(k)} \mid$ in terms of $\mid I^{(k-1)} \mid$. There is a non-zero probability that an event in $I^{(k-1)}$ seeds a rollback tree. Let us give a bound for the number of events in this rollback tree.

By the argument dealing with step 3c above, the total number of events created by a rollback at any node in the rollback tree is bounded by a random variable with an exponential tail, say $V$, which is independent of the initial rollback size, of $k$ and of $N$. The number of nodes involved in the rollback tree is bounded by the tree (Branching Random Walk) $Br$ with initial rollback size $B$. We have

$$\mid I^{(k+1)} \mid \leq \sum_{j=1}^{|I^{(k)}|} \left( \sum_{l=1}^{Br_j} V_{kj}(l) \right) \triangleq \sum_{j=1}^{|I^{(k)}|} f_{kj} \tag{9}$$

where $V_{kj}(l)$ is distributed as $V$, $kj$ identify an event $e$ in $I^{(k)}$ which started a rollback tree, and $l$ identifies a node on this tree. Here $f_{kj}$ represents the total number of events in the rollback tree (if any) seeded by event $j$ of $I^{(k)}$. Thus we see that the size of $I^{(k)}$ is bounded by a standard Branching Process, with $\mid I^{(0)} \mid = CN$. We now proceed to prove the existence of moments for some of the Branching Processes. We shall use the following technical results.

**Lemma 1.** *Let the positive constants $g_i$ sum to one. Then for any random variables $\{\xi_i\}$*

$$P\left(\sum_i \xi_i \geq t\right) \leq \sum_i P(\xi_i \geq t g_i)$$

**Corollary 1.** *If* $E\left(|\xi_i|^{1+\beta}\right) \triangleq \zeta_i < \infty$ *then*

$$P\left(\sum_i \xi_i > t\right) \leq \sum_i \frac{\zeta_i}{(tg_i)^{(1+\beta)}}$$

The proof of Lemma 1 is immediate. The Corollary then follows using Chebycheff's inequality.

**Lemma 2 [A].** *For any independent random variables* $A_i \geq 0$, *let* $S = \sum_{i=1}^n A_i$. *Then for any* $0 < \beta \leq 1$,

$$E\left(S^{1+\beta}\right) \leq (ES)^{1+\beta} + \sum_{i=1}^n E\left(A_i^{1+\beta}\right)$$

**Proof.** Since for any positive $a, b$, $(a+b)^\beta \leq a^\beta + b^\beta$ we find

$$\begin{aligned} E\left(S^{1+\beta}\right) = E\left(S\, S^\beta\right) \leq & E\left(\sum_{i=1}^n A_i \left[\left(\sum_{j\neq i} A_j\right)^\beta + A_i^\beta\right]\right) \\ \leq & \left(E\sum_{i=1}^n A_i\right) E\left(\sum_{i=1}^n A_i\right)^\beta + E\left(\sum_{i=1}^n A_i^{1+\beta}\right) \end{aligned}$$

by independence and non-negativity of $A_i$. Hence by Jensen's inequality applied to the *concave* function $x^\beta$ the Lemma is proven. ■

**Corollary 2.** *Let* $n$ *be a non-negative integer valued random variable which is measurable on a sigma-field* $\mathcal{F}$. *Let* $A_i > 0$ *be random variables which are independent, and independent of* $\mathcal{F}$. *let* $S = \sum_{i=1}^n A_i$. *Then, for any* $0 < \beta \leq 1$,

$$E\left(S^{1+\beta} \mid \mathcal{F}\right) \leq (E(S \mid \mathcal{F}))^{1+\beta} + \sum_{i=1}^n E\left(A_i^{1+\beta}\right)$$

Let $K(n)$ be a Branching Process with Branching described by the i.i.d. random variables $\{Y_{nj}\}$ with distribution $\{q_i\}$, with $EY_{nj} = m$ and $E\left(Y_{nj}^{1+\beta}\right) = c_2 < \infty$.

**Lemma 3.** *Let* $k(0) = M$. *There exists a constant* $c_1$ *such that*

$$E\left(k(n)^{1+\beta}\right) \leq c_1 \left(Mm^n\right)^{1+\beta} \quad \text{if } m > 1$$

$$E\left(k(n)^{1+\beta}\right) \leq c_1 m^n M^{1+\beta} \quad \text{if } m \leq 1$$

Comment: The condition that a $1+\beta$ moment exist is a touch stronger than the familiar XlogX condition used in most Branching Process results. We don't know if we are proving the weakest possible form of our theory, but having a $1+\beta$ moment is not very much worse than XlogX in any case.

**Proof:** Use Corollary 2 to see that

$$E\left(k(n)^{1+\beta} \mid k(n-1)\right) \leq (mk(n-1))^{1+\beta} + k(n-1)E\left(Y_{nj}^{1+\beta}\right)$$

Hence

$$E\left(k(n)^{1+\beta}\right) \leq m^{1+\beta} E\left(k(n-1)^{1+\beta}\right) + m^{n-1} c_2$$

This easily proves Lemma 3. ■

**Lemma 4.** *Under Assumption 4 , there exists an $\epsilon > 0$ such that*

$$E\left(f_{kj}^{1+\epsilon}\right) < \infty$$

**Proof.** Recall that $z(n)$ is the number of live nodes in the Branching Random Walk at the $n^{\text{th}}$ stage. We take the initial size $X_0 = B$. Then

$$E\left(z(n)^{1+\beta}\right) = E\left(\left(\sum_{i=1}^{k}(n)\mathbf{1}(x(n,i) > 0)\right)^{1+\beta}\right)$$

$$\leq E\left(k(n)^{\beta} \sum_{i=1}^{k(n)} \mathbf{1}(x(n,i) > 0)\right)$$

by a slight extension of [LSWb, Lemma 1 and the Proof of Proposition 1]

$$= E\left(k(n)^{1+\beta}\right) c_3 e^{-nh}$$

and by Corollary 2,

$$\leq c_4 b^{(1+\beta)n} e^{-nh} \ .$$

Since $be^{-h} < 1$, if $\beta$ is chosen small enough, then

$$b^{1+\beta} e^{-h} < 1 \ .$$

Hence for some $r_1 < 1$, $E\left(z(n)^{1+\beta}\right) \leq c_5 r_1^n$.

Next we treat $Br_j$. By definition,

$$Br_j \triangleq \sum_{n=0}^{\infty} z(n)$$

if the appropriate event $e$ seeds a tree, and is zero otherwise. Hence using Lemma 2 and Corollary 2,

$$E\left(Br_j^{(1+\beta)}\right) \leq (E(Br_j))^{(1+\beta)} + P(\text{event } e \text{ seeds a rollback tree}) \sum_{n=0}^{\infty} E\left(z(n)^{(1+\beta)}\right)$$

But note that

$$E(Br_j) \leq P(\text{event } e \text{ seeds a rollback tree}) E(\text{size of tree } \mid e \text{ seeds a tree}) .$$

By Assumption **2** , we can make $P(\text{event } e \text{ seeds a rollback tree})$ as small as desired, uniformly in $j$ and $N$, by a proper choice of $d(i,j)$. Thus Theorem 1 implies that

$$E\left(Br_j^{(1+\beta)}\right) \leq c_6 P(\text{event } e \text{ seeds a rollback tree})$$

Now we use Corollary 2 to bound $f_{kj}$ defined in (9).

$$\begin{aligned} E\left(f_{kj}^{(1+\beta)} \mid Br_j\right) &\leq \left(E\left(\sum_l^{Br_j} V_{kj}(l)\right)\right)^{1+\beta} + \sum_l^{Br_j} E\left(V_{kj}(l)^{1+\beta}\right) \\ &= ((EV_{kj}(l))Br_j)^{(1+\beta)} + Br_j E\left(V_{kj}(l)^{(1+\beta)}\right) . \end{aligned}$$

So, taking expectations,

$$E\left(f_{kj}^{(1+\beta)}\right) \leq c_7\left(EBr_j^{(1+\beta)} + EBr_j\right) \leq c_8 P(\text{event } e \text{ seeds a rollback tree}) .$$

Define $m_0 \triangleq Ef_{kj}$, and $m_1 \triangleq E\left(f_{kj}^{(1+\beta)}\right)$, and choose $d(i,j)$ so that $m_0 < 1$ and $m_1 < 1$ (this can also be accomplished by choosing $B$ small enough, see Assumptions **2** and **3**). Let $Y$ be the number of cycles in Filtered Rollback and let $W$ be the total number of events. Assumption **6** gives, for some $c_9 < \infty$ and some $0 < \rho_1 < 1$

$$P(Y > j + rc_9 \mid W = rN) \leq \rho_1^j .$$

We now have

$$P(W > rN) = P\left(\sum_k \mid I^{(k)} \mid > rN\right) \leq \sum_k P\left(\mid I^{(k)} \mid > rN\frac{g^k}{1-g}\right)$$

where $g$ is chosen to satisfy $1 > g^{(1+\beta)} > m_1$. Using Lemmas 3 and 4 we can find $c_{10}$ so that $E\left(\mid I^{(k)} \mid^{(1+\beta)}\right) \leq c_{10}N^{(1+\beta)}m_1^k$. Corollary 1 now gives

$$P(W > rN) \leq \sum_k \frac{(1-g)c_{10}N^{(1+\beta)}m_1^k}{(rNg^k)^{(1+\beta)}} \leq \frac{c_{11}}{r^{(1+\beta)}} .$$

Finally,

$$\begin{aligned}
P(Y > j) &\leq \sum_{t=1}^{\frac{jN}{2c_9}} P\left(Y > j - \frac{c_9 t}{N} + \frac{c_9 t}{N} \mid W = t\right) P(W = t) + P\left(W > \frac{jN}{2c_9}\right) \\
&\leq \sum_{t=1}^{\frac{jN}{2c_9}} \rho^{(j-\frac{tc_9}{N})} P(W = t) + \frac{c_{11}}{(\frac{j}{2c_9})^{1+\beta}} \\
&\leq \max_{t \leq \frac{jN}{2c_9}} \rho^{(j-\frac{tc_9}{N})} \sum_{t=1}^{\frac{jN}{2c_9}} P(W = t) + \frac{c_{11}}{(\frac{j}{2c_9})^{1+\beta}} \\
&\leq \rho^{\frac{jN}{2c_9}} + \frac{c_{11}}{(\frac{j}{2c_9})^{1+\beta}}
\end{aligned}$$

Hence

$$\sum_k k^{(1+\epsilon)} P(Y = k) \leq c_{12} \sum_k k^{\epsilon} P(Y \geq k) < \infty$$

provided $\epsilon < \beta$. In Lemma 5 below we show that this implies that for some $\delta > 0$,

$$\sum_k kP(Y = k)^{(1-\delta)} < \infty .$$

**Lemma 5.** *If $\sum_k k^{(1+\epsilon)} p_k < \infty$ for $0 < \epsilon < 1$, then for any $0 < \delta < \frac{\epsilon}{3}$ we have*

$$\sum_k kp_k^{(1-\delta)} < \infty .$$

**Proof**: Estimate separately for those $k$ such that $p_k \geq k^{-3}$ or $p_k < k^{-3}$. ■

With this we conclude the proof as follows. Since $U_i$ satisfies, for some $C_1, C_2, r$

$$P(U_i \geq C_1 \log N + t) \leq C_2 e^{-rt}$$

we obtain, for any $A > 1$,

$$\left(E(U_i^A)\right)^{1/A} \leq C(A) \log N$$

Hence

$$E\left(\sum_{i=1}^{Y} U_i\right) = E\left(\sum_{k=1}^{\infty}\sum_{i=1}^{k} U_i \mathbf{1}(Y=k)\right) = \sum_{k=1}^{\infty}\sum_{i=1}^{k} E\left(U_i \mathbf{1}(Y=k)\right)$$

and by Holder's inequality

$$\leq \sum_{k=1}^{\infty} k \left(E(U_i^A)\right)^{1/A} \left(P(Y=k)\right)^{1/Z}$$

where $\frac{1}{A} + \frac{1}{Z} = 1$. Take $\frac{1}{Z} = 1 - \delta$, so $A = \frac{1}{\delta}$, and we obtain by Lemma 5

$$E\left(\sum_{i=1}^{Y} U_i\right) \leq C(A) \log N \sum_{k=1}^{\infty} k \left(P(Y=k)\right)^{(1-\delta)} = c_{13} \log N \ .$$

This establishes the efficiency result, Theorem 4. ■