Proc. 16th Workshop on Parallel and Distributed Simulation, Washington, DC, May 2002, IEEE CS Press, pp. 153-162

Lookback: A New Way of Exploiting Parallelism in Discrete Event Simulation

Gilbert Chen Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street, Troy, NY 12180-3590
{cheng3,szymansk}@cs.rpi.edu

Abstract

Lookback is defined as the ability of a logical process to change its past locally (without involving other logical processes). Logical processes with lookback are able to process out-of-timestamp order events, enabling new synchronization protocols for the parallel discrete event simulation. Two of such protocols, LB-GVT (LookBack-Global Virtual Time) and LB-EIT (LookBack-Earliest Input Time), are presented and their performance on the Closed Queuing Network (CQN) simulation is discussed. We show that lookback can be used to reduce the rollback frequency in optimistic simulations. We also discuss in detail the relation between lookahead and lookback. Finally, we demonstrate that lookback allows conservative simulations to circumvent the speedup limit imposed by the critical path.

1. Introduction

Researchers have long realized that Parallel Discrete Event Simulation (PDES) is an effective approach to simulating large-scale complex systems. Research on PDES has been going on for more than twenty years. The main difficulty in this area is to achieve high efficiency of parallel execution while preserving the causality order between events for the simulation carried out on multiple processors. The logical process paradigm, widely used in the PDES community, assures that no causality errors will occur if each logical process adheres to the *local causality constraint* [1], i.e., if each logical process executes its events in non-decreasing timestamp order. Therefore, to preserve the causality order, it is sufficient, though not always necessary, that each logical process finds and executes the future event with the smallest timestamp.

Research on PDES has been largely dominated by the studies of conservative [2,3] and optimistic protocols [4], and comparison of their performance. Unfortunately, both types of protocols have their strengths and weaknesses. Efficiency of conservative protocols in parallel execution is limited by the amount of lookahead in the simulation model, which in each logical process is equal to the difference between its Earliest Output Time (EOT) and its Earliest Input Time (EIT) [5]. Both EIT and EOT are known exactly only during run-time and in many real world applications, deriving bounds for the difference between these two values, which will define useable lookahead, is difficult to do. Additionally, null messages required to collaboratively advance the simulation clock in conservative protocols often incur significant overhead. As a result, parallelized execution may be slower than even the sequential one. On the other hand, optimistic protocols do not depend on lookahead and null messages. However, state saving usually requires storing and accessing large amounts of memory. This negatively impacts the speed of execution because of the relatively slow improvement in the memory access speed within the current VLSI technology. The handling of anti-messages complicates the simulation model development. Furthermore, as Nicol and Liu have pointed out in [6], optimistic models may exhibit unexpected behavior caused by inconsistent messages resulting from rollback inconsistencies and stale states.

In this paper, we show that lookback, found in many simulation applications, provides yet another source of parallelism both for conservative and optimistic protocols. Lookback can reduce the number of rollbacks in optimistic simulations, because a logical process may simply process an out-of-timestamp order event without rolling back other events. Lookback also supports new conservative protocols, some of which may no longer

depend on lookahead. The most important theoretical consequence is that lookback-based conservative protocol execution time may be lower than the bound given by the critical times of events.

2. What is Lookback

Informally, lookback is defined as the ability of a logical process to execute out-of-timestamp order events, which are often referred to as *stragglers,* without impacting states of other logical processes. We first give the formal definition of lookback and other related terms.

Definition 1: If a logical process at simulation time *T* can execute correctly, without sending out anti-messages, any received event with timestamp between *T-L* and *T*, then the time window *[T-L, T]* is said to be the *lookback window* of the logical process at time *T*. The lower end of the lookback window, *T-L*, is referred to as the *virtual lookback time*. The procedure used to process events falling into the lookback window is called the *lookback procedure*. For convenience, the *lookback window* is sometime abbreviated to *lookback*. Finally, for any future event *E* at the given simulation time *T, LB(E,T)* denotes the function that defines the value that the virtual lookback time would have if the event *E* were executed at time *T*.

As defined, lookback is the property of lookback procedure and the partitioning of the simulation into logical processes. However our experience shows that in many cases it is relatively easy to see what effect each event has on lookback and what processing needs to be done for each event that falls into the lookback window. As a simple example, consider a logical process that simulates a server with a First-Come-First-Serve (FCFS) queue receiving tasks for processing and sending processed tasks to other logical processes. An arrival of a task in the logical process does not change the lookback, if we place the arriving task in the queue according to its simulation arrival time. If the tasks arrive in the order of their simulation arrival times, then merely placing them in the FCFS order in the queue would suffice. On the other hand, the departure of the task from the server sets the virtual lookback time to the simulation arrival time of the departing task, let's call it *T*. The reason is simple, a newly arriving task with the simulation arrival time smaller than *T,* would need to be processed before the departing tasks, changing its departure time. Since the departing task carries its departure time to some other logical process, the change required in such a case is no longer local. Similar analysis of the processing of events within the lookback window and of an impact of each event on the virtual lookback time allows the simulation designer to define the correct lookback procedure and function *LB(E,T)*.

To avoid any causality error or rollbacks at other logical processors, each logical process must maintain a virtual lookback time that is less than or equal to the timestamp of any future event. Under such circumstances, any received event can be successfully processed by either the regular event handler or the lookback procedure. Therefore, we define the lookback constraint as follows.

Definition 2: A logical process obeys the *lookback constraint* if and only if after processing each event, the virtual lookback time is smaller or equal to the minimum timestamp value of all unprocessed events of this logical process (including those that will arrive at the logical process later). The *Minimum Timestamp* of all unprocessed events at the simulation time T is denoted as MT(T).

The lookback constraint is basically a relaxation of the local causality constraint given by Fujimoto [1]. Adherence to the lookback constraint enables a new kind of conservative protocols for the parallel discrete event simulation. We give the description of the most general protocol below using the following notation. *T* denotes the global simulation time, U is the set of all unprocessed events on the given logical process, $S \subseteq U$ denotes a subset of *U* and *LBTS(T)* stands for the lower bound of *MT(T)*.

```
while (the termination condition is not met)
    E(T)={∀e∈S : LB(e,T) ≤ LBST(T)},
    if ( E(T) is nonempty)
        while ( E(T) is nonempty )
            remove an event from E(T) and process it
        end while
    else
        recompute LBTS(T)
    end if
end while
```

This general lookback-based protocol first constructs a subset *E(T)* of all local events eligible for execution. It then repeatedly removes an event from *E(T)* and *executes it*. The *LBTS* is recomputed only when *E(T)* is empty, so no event has been processed.

Theoretically, using all unprocessed events and exact value of *MT(T*) makes the protocol deadlock free.. However, the efficiency of creating the subset *E(T)* is important for overall efficiency of simulation because this operation is repeated for each iteration. *LBTS(T)* also may be costly to evaluate exactly, and often its lower bound, such as GVT or EIT is maintained in the system*)*.

Hence, in this paper, we consider variants of this general protocol in which only a single event is in set S. One way to do it is to select S containing only the event with the smallest timestamp. Such a solution minimizes the amount of processing by minimizing the number of invocations of the lookback procedure which is usually more expensive that the regular event procedure. Another possibility is to order the future event list by the virtual lookback time of events and select the event with the smallest virtual lookback time as the only member of S. This solution will work well for simulations whose virtual lookback time is independent of the simulation time. It increases the parallelism of the simulation and minimizes the number of times LBTS is recomputed when the earliest event cannot be processed.

We have implemented two variants of the general lookback-based protocol, which always select the earliest event as the only member of *E(T)*. The algorithm is given below. These two variants differ in the estimations of the LBTS: one is using GVT (Global Virtual Time) and the other is based on EIT (Earliest Input Time). We abbreviate the two variants of the protocol as LB-GVT and LB-EIT, respectively.

```
while (the termination condition is not met)
    select the earliest event e from the local event
list
    if LB(e,T) is not greater than GVT(or EIT)
        remove e from the local event list
        process e
    else
        recompute GVT(or EIT)
    end if
end while
```

The LB-GVT protocol keeps a global estimation of the LBTS as equal to GVT. Hence, it does not take into account the interconnections between logical processes. As a result GVT is a crude estimation of the LBTS. In contrast, the LB-EIT protocol requires that each logical process maintain its own estimation of the LBTS based on the topology of the interconnections, thus, helping to improve the accuracy of the estimation. Similar to the Null Message protocol, however, LB-EIT is prone to deadlock, when a cycle of logical processes is formed where every logical process assumes the earliest event will come from others. Therefore, the LB-EIT protocol still requires the knowledge of lookahead to break up potential deadlocks.

Theorem 1: LB-GVT is deadlock-free, if the GVT is at least accurate at the time of recomputation, i.e., if at the time of recomputation the GVT is equal to the smallest timestamp of all unprocessed events in the simulation.

Proof: We will proceed by induction over the event execution. At simulation time 0, virtual lookback time, and *gvt* are equal, so it holds that $lvt \leq gvt$. Denote the timestamp of the earliest unprocessed event(s) in the system by *t*. Processing an event with timestamp *t* cannot cause a causality error, because by the inductive assumption, the lookback constraint holds before such an event arrives, which implies the virtual lookback time $vlt \leq gvt=t$. We need to show that the lookback constraint still holds after processing such an event. The fact that the lookback is non-negative implies that after the processing, $vlt \leq t$. Since $t=gvt$, we have $vlt \leq t=gvt$. The lookback constraint, which requires $vlt \leq gvt$, is not affected and any event with the smallest timestamp *t* can be safely processed. Therefore, a deadlock cannot arise. □

3. Lookback in Simulation Models

We have introduced the concept of a lookback and sketched two synchronization algorithms for Parallel Discrete Event Simulation (PDES) that rely upon the existence of lookback. The key question related to the usefulness of these two synchronization algorithms is, how common is the lookback in real world simulations? We try to answer this question by presenting two arguments. First, we will demonstrate that lookback, the ability to change the past, is not limited to the models where events are independent of each other. Second, we will show that lookback is more commonly observed than lookahead.

3.1 Lookback and Out-of-Timestamp Order Execution in Optimistic Protocols

The most perfect lookback might be that of linear systems. It has been demonstrated that the superposition property of a linear system can be used to process a straggler without a rollback [7]. Instead, a new copy of the current system is initialized with all state variables set to zero. This newly created copy is run with the straggler as the input from the timestamp of the straggler to the current simulation time. Finally, the correct state information is created by summing up the state variables in the original system with those in the new copy. The interesting fact is that we are able to correct the values of the state variables at the current simulation time without ever recreating their values at the simulation time when the straggler is bound to execute.

In the PDES community, it has been noticed by many researchers that in some special circumstances events can be processed in out-of-timestamp order, in some special

circumstances, for instance, when events are independent of each others.

Query events, proposed by Sokol [8], are the simplest form of independent events. They do not change the internal state of the logical process, and therefore can be safely processed even if the local simulation time is larger than their simulation time (i.e., when they are formally stragglers), by simply reading the state history, instead of rolling back the entire logical process. Other events are then called side-effecting events and require a rollback if they are received after the local simulation time progressed beyond the time at which they were supposed to be scheduled.

Rollback relaxation is a similar technique [9]. Logical processes can be classified into memoried processes and memoryless processes. In the former, the output messages are computed as a function of both input messages and internal state variables, while in the later only the input messages are used. Consequently, in memoryless processes rollback relaxation is able to process stragglers without state-saving.

Weak Causality [10] is a formal notion defining the event dependency which simplifies the detection of independent events. Traditionally, causality is defined in terms of the happened-before relation. The Weak Causality relation is based on the conflicts created by events that operate on the same data, instead of on the timestamps of events.

Leong and Agrawal [11] have proposed a semantics-based optimistic protocol that exploits the semantics of messages to avoid certain types of rollback. For example, the *commutativity* relation determines if two events can be executed out of order, while the *invalidated-by* relation determines whether a processed event should be rolled back due to the arrival of another event with a smaller timestamp. However, even for sets, the data structure for which relations between different types of message were discussed in their paper, this protocol is complex. Their work cannot be easily extended to other general simulation models.

All previous attempts trying to exploit the independence between events have one thing in common: upon arrival of a straggler, they first check by some means whether the straggler can be safely handled. If the straggler can be processed, the *same* procedure that processes positive messages is invoked. Otherwise, a rollback-recovery procedure is invoked. In a word, the event independence is used only to reduce the number of rollbacks.

In contrast, we propose that lookback based protocols handle stragglers in a different way. First, the logical process will determine whether a straggler can be safely processed according to the notion of lookback. A *different* procedure, the lookback procedure, will be invoked to process stragglers whose timestamp is within the lookback window. The lookback procedure will effectively carry out two operations. It first tries to repair the damage caused by the wrong order of event execution. Second, it processes the straggler based on the state that has been repaired. Not only does our approach reduce the number of rollbacks, it may also completely eliminate rollbacks. By adhering to the lookback constraint, the protocol guarantees that all stragglers will be correctly processed by the lookback procedure, thus eliminating the need for the rollback and recovery procedure.

When the lookback procedure is used to deal with independent events, it is apparent that the repair operation is no longer needed. How well it will handle dependent events depends on its ability to repair the damage. It is intuitive that any changes made to the local state of the logical process can be recovered by some means, but the sent messages can be only cancelled out by sending anti-messages. The following definition specifies a universal lookback procedure that is able, although rarely in the most efficient way, to correctly process stragglers except in the case where new events generated in the erroneous computation have been delivered to other logical processes.

Definition 3. A *universal lookback procedure* requires every logical process to save the list of processed events and every change made to the state by each event. Upon arrival of a straggler, it rolls back all processed events with a timestamp larger than the timestamp of the straggler, in decreasing timestamp order. Then, it processes the straggler, and finally re-execute the events that have been rolled back, in increasing timestamp order.

Clearly, the lookback-based protocol adopting the universal lookback procedure belongs to the class of *local rollback* mechanisms [12]. Anti-messages are strictly avoided but local state-saving is still required. This also implies that the aforementioned lookback-based protocols unify two previously known synchronization techniques, out-of-timestamp order execution and local rollback. When out-of-timestamp order execution of events is allowed, the discrepancy between the simulation times in different processors can be smoothed out by the logical processes on the boundary of processing elements, by means of local rollback. All other logical processes observe no stragglers or anti-messages. This suggests that lookback-based protocols allow for aggressiveness, but not risk [13]. Two such protocols previously known, SRADS with local rollback [12] and Breathing Time Window

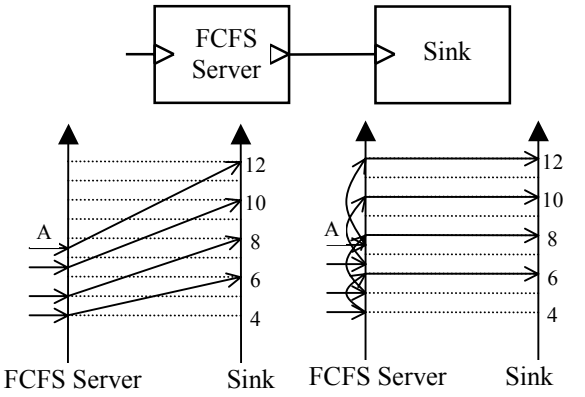
[14], must require logical processes to work collaboratively, by exchanging event information, to avoid sending any erroneous message. In lookback-based protocols, however, each logical process can determine alone whether an event can be sent out.

The universal lookback procedure deals with the worst case of event dependence. It is possible that no state, or only a small portion of the entire state need to be saved to enable a logical process to process the stragglers, as is the case of the Closed Queuing Network simulation that we will discuss later. In some models, when the lookback procedure is used to reduce the number of rollbacks in optimistic simulations, only an extra variable might be needed to store either the value of the virtual lookback time or the size of the lookback window.

3.2 Lookback and Lookahead

To understand the relationship between lookback and lookahead, we must first turn our attention to two different scheduling mechanisms for delivering inter-logical process events. The distinction between *eager* and *lazy scheduling* was motivated by Fujimoto's notion of eager and lazy server [15]. In eager scheduling, an inter-logical process event is sent out immediately when it is generated. Such an event usually contains a timestamp larger than the current simulation time of the sender. In lazy scheduling, an event is sent out only when the simulation clock reaches the timestamp of the event. Therefore the timestamp of any outgoing event is always equal to the current simulation time of the sender.

In Figure 1, we consider an example of an FCFS server using these two scheduling schemes. Suppose a sequence of tasks arrive at times 4, 5, 6.5 and 7.5. The FCFS server has a constant service time of 2 time units for all tasks. With eager scheduling, the departure event for each task can be scheduled and sent to the sink as soon as the task arrives, because of the nature of the FCFS server. Here, the lookahead is exploited to the maximum degree. For instance, when the task A arrives at time 7.5, it can be immediately calculated that it will depart at time 12, so the lookahead is 12-7.5=4.5. On the other hand, lookback does not exist, because at the moment of the event departure from the server, the effect of the processed event becomes permanent and cannot be recovered.



(a) Eager Scheduling (b) Lazy Scheduling

Figure 1. Eager Scheduling versus Lazy Scheduling

In lazy scheduling, the departure event is not immediately sent to the sink. Instead, it is held in the server's event list until the current simulation time is equal to its timestamp. This delay gives the logical process a chance to retract a departing event in case an arrival event with a timestamp smaller than the arrival timestamp of the departing task is received later. In such a case, the previously scheduled departure time should be changed after scheduling the departure event of the newly arriving task first. Lookback is the largest if the scheduled departure event is sent out as late as possible. Therefore, *lookback-based protocols prefer lazy scheduling*.

How much is the lookback in an FCFS server? We can see that when the task A leaves the server at time 12, the virtual lookback time is changed to the arrival time of the task A, which is 7.5. Indeed, any task arriving at the simulation time later than 7.5 can be simply inserted into the waiting queue. However, a task scheduled to arrive earlier than 7.5 would force a causality error, because it is this task that must leave the server at simulation time 12, not task A. The lookback window size is therefore 12-7.5=4.5. Interestingly, the lookback under lazy scheduling is equal to the lookahead under eager scheduling in this case. This is by no means a coincidence, but rather implies that there is a relation between lookback and lookahead, which is revealed in the following theorems.

Theorem 2: If under eager scheduling, a simulation contains a lookahead of L at simulation time T after event execution, then under lazy scheduling this simulation contains a lookback of at least L at time $T+L$ before execution of events that sends out messages.

Proof: Under eager scheduling, a logical process is said to contain a lookahead of L at simulation time T if it can

only schedule new events with timestamp of at least $T+L$. Consider this simulation under lazy scheduling using a lookback-based protocol. Suppose at simulation time $T+L$ a straggler arrives with a timestamp within $[T, T+L]$. Since this straggler arrives no earlier than time T, any message that it produces must contain events with the simulation time larger or equal to lookahead at time T, which is $T+L$, so these messages will be placed on the future event list. Any message that leaves the logical process within time $[T,T+L)$ under lazy scheduling must have been produced by an event at the simulation time less than T, otherwise the lookahead could not have been L, so such a message cannot be affected by this straggler. Hence, no messages will be sent out during processing of such stragglers, and if any straggler affects a message scheduled at exactly time T, the logical process still has a chance to cancel the message, because according to the condition given by the theorem such a message should have not been executed. The associated operations of the universal lookback procedure, which only change the state of the logical process, are always valid. □

Now we know that lookback is at least as common as lookahead. Whenever lookahead exists, lookback is also available. Can lookback exist without lookahead? The following theorem answers this question positively.

Theorem 3: Lookback may exist even when lookahead is zero.

Proof: We give a simple example to support this claim. Assume a zero delay logical process whose only function is to copy any received message from an input port to an output port. Apparently there is no lookahead in the logical process, because messages can arrive at any time and they leave immediately. The lookback, however, is infinite. The logical process can accept messages with any timestamp no matter what is its current simulation time. □

Theorem 2 and Theorem 3 together tell us that existence of lookback could be more commonly observed than that of lookahead. One interesting question is, under what circumstances will a lookback transform into a lookahead? The theorem below answers this question.

Theorem 4: Assume a logical process has a lookback of L at simulation time T, again under lazy scheduling, and there is an outgoing message timestamped with time T. If the underlying simulation model guarantees that all other messages sent out later will contain a larger timestamp, then at simulation time $T-L$ the logical process has a lookahead of L.

Proof: Denote by t the timestamp of the event that schedules the outgoing message. By definition of lookback at time T, $t \leq T-L$, otherwise an event with timestamp $t>T-L$ would affect the outgoing of the message at time T, which violates the assumption that the lookback is L. Let us consider the logical process at the simulation time $T-L$. If eager scheduling is used, the event at time t must have already been processed, with the outgoing message scheduled. Since no messages can be sent in the window $[T-L, T]$, it appears that $EOT=T$, i.e., the next output message is at T, therefore the lookahead at $T-L$ is exactly L. □

Theorem 4 explains why in the FCFS server model the lookback is no greater than the lookahead. On the other hand, lookback and lookahead do not always exclude each other. With lazy scheduling, a logical process can still inform others of its next output time without sending the actual event, as stated by the theorem below.

Theorem 5: Under lazy scheduling, if at the current simulation time T a logical process can predict an *opaque period* [16] of L simulation time units, during which no message will be sent, then the lookahead at time T is equal to L and the lookback window at time $T+L$ is at least L.

Proof: By definition, lookahead at time T is L, because the Earliest Output Time is at least $T+L$. Also, when the simulation clock advances to $T+L$, under lazy scheduling, the process can, in the worst case using the universal lookback procedure, execute any events received within the time $[T, T+L]$, as described in the proof of Theorem 2. Thus, the opaque period L becomes part of the lookback window. □

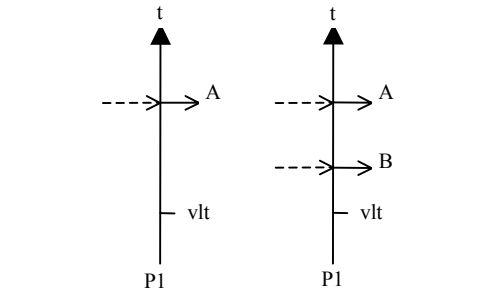
Opaque periods, which are in essence promises made by a logical process not to send messages to neighbors within a certain time window, have been known to be a primary source for lookahead. Theorem 5 states that an opaque period also implies at least the same amount of lookback. This may establish a new mechanism to exploit both lookback and lookahead at the same time.

To summarize, to some extent, lookback is a dual of lookahead: lookahead is the ability to predict the future, while the lookback is the ability to change the past. At first glance they seem to be completely unrelated. Nevertheless, when we are able to advance the simulation clock more aggressively, the future becomes the past. The relations between lookahead and lookback dictated by the above three theorems are then easy to understand.

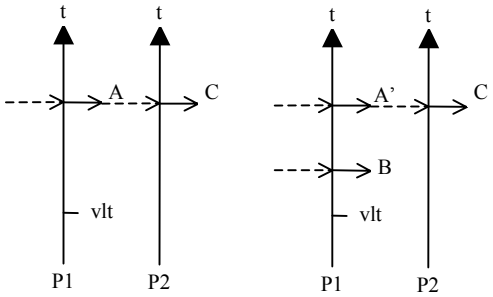
3.3 Lookback and Super-Criticality

Are lookback-based protocols, namely LB-GVT and LB-EIT, conservative or optimistic? This is the distinction we must first make in order to study the super-criticality of lookback-based protocols. The use of the universal

lookback procedure makes them more or less like optimistic, but since there exist models in which no state-saving is required to exploit lookback, we argue that the optimism of lookback-based protocols varies with the selected model. At least for those models requiring no state-saving, the lookback-based synchronization should be undoubtedly characterized as conservative.



(a) Independent Events A and B



(b) Independent Events B and C

Figure 2. Supercriticality and Independent Events

There is a unanimous opinion that conservative protocols cannot beat the critical path bound on execution time [17]. However, we will show that this is not true for lookback-based protocols. For the convenience of discussion, we denote the starting time of an event *e* by *start(e)*, the completion time by *complete(e)*, and the critical time by *critical(e)*. There are two cases in which lookback-based protocols can produce super-critical speedup.

An example of the fist case is depicted in Figure 2(a), in which we assume that the event A executes first. Later, another event B is received with a timestamp smaller than that of A but greater than the virtual lookback time. The event B can then be processed safely by the lookback procedure. For this case, assume that these two events are independent of each other. Thus, when processing the event B, the lookback procedure does not need to correct the logical process state reached after the execution of the event A. Apparently, the event A completes even before the event B starts execution. For simplicity, assume that the event B is on the critical path, i.e., *complete(B)=critical(B)*, and therefore *complete(A)<start(B)<complete(B)=critical(B)< critical(A)*, so *complete(A)<critical(A)*, which is the sufficient condition for super-criticality given in [18].

In the other case depicted by Figure 2(b), the two events A and B are no longer independent. Rather, after processing the straggler B, we have to repair the incorrect state produced by the first execution of the event A. Therefore *complete(A)>complete(B)*. But the event C created by the first execution of the event A is not affected. We say that the event C is independent of the event B for this reason. Again, if we assume that the event A is on the critical path, then *complete(C)< complete(B)<complete(A)=critical(A)<critical(C)*, or *complete(C)<critical(C)*, which proves the event C is a super-critical event. It should be noted that the lookback impact of event A must be based in this case on the non-atomic semantics of this event. Clearly, the triggered event C cannot depend on the local state created by the execution of event A. Making such a distinction in practice is difficult albeit not impossible.

Jefferson and Reiher have proven that all conservative mechanisms are bound by the critical times of events [17] but under the assumption that all correct conservative simulation mechanisms must use *elementary scheduling*. In it, for all pairs of committed events *e* and *e'*, whenever *e* is either the predecessor of *e'* or the antecedent of *e'*, then *complete(e)≤ start(e')*. The authors claim that otherwise the simulation might be incorrect.

However, neither do they consider mechanisms that ensure that stragglers can always be executed correctly, nor mechanisms other than guessing that can allow the logical process to start execution before its antecedent. The lookback-based protocol guarantees that once an event is about to execute, all predecessors of this event, if there are any, can also be processed correctly. If it cannot make such a commitment, the event processing has to be delayed. Also, with lookback, events are no longer atomic [19]. As in the example of Figure 2(b), the event A is executed two times, one by the regular event procedure and the other by the lookback procedure, and the event C, produced by the first execution of the event A, is guaranteed to be correct by the notion of the lookback (one example could be the event triggered by the arrival of event A on logical process P1, which is not changed by earlier or later arrivals of other events on logical process P1). If it were not, the event A would fail to be processed in the first execution and would delay its first completion, because no sent out event can be rolled back.

It is worth to note that lookback is limited by the message dependence introduced in [19]. Informally, a message, created by an event *e*, is dependent on an event set *E* consisting of events that influence *e*, if this message

would be different if any event in the set *E* had not been executed. In this case, the event *e* cannot be executed before any event in the set *E*, because it would send out an incorrect message which cannot be cancelled.

Definition 4: The *impact time* of a message is maximum timestamp in the event set E on which this message is dependent (as defined above).

The impact time of a message gives an upper bound of the lookback one can exploit. This can serve as a guideline in determining whether the lookback-based protocols are applicable, and how much lookback can be obtained.

It is also worth to note that lookback-based protocols and lazy cancellation [20] bear some resemblance. Lazy cancellation determines whether or not an anti-message is necessary by comparing the positive message newly generated by the rollback and recovery procedure with the positive message that has already been sent out. If they are the same, there is no need to send the anti-message. In contrast, when lookback is applied to optimistic simulation, it determines the necessity of an anti-message by comparing the timestamp of the straggler with the virtual lookback time. If the straggler timestamp is within lookback window, then the lookback procedure can handle the straggler and no anti-messages need to be sent. Otherwise, the rollback and recovery procedure is invoked and the corresponding anti-messages are sent. To make this decision, lookback uses only one floating-point number comparison, avoiding the high overhead of message comparison associated with lazy cancellation.

4. Closed Queuing Network Simulation

A Closed Queuing Network (CQN) consists of switches and FCFS servers (Figure 3). Each task traveling through one of the FCFS server tandems will eventually reach a switch. The switch will then dispatch the task to one of the tandems randomly.

Lookback-based protocols are well suited for the CQN simulation. The lookback of the switch is infinite. The virtual lookback time of the FCFS server is always equal to the receive time of the last task leaving the server. Any task received with a timestamp smaller than that of the current task in service must preempt the later. All those with a timestamp greater than or equal to the virtual lookback time can be correctly inserted into the waiting queue at positions defined by their timestamps. Tasks with timestamp smaller than the virtual lookback time will trigger a causality error. But such errors are preventable; when a task is about to leave the server, its receive time must be checked against the LBTS. If the receive time is greater than the LBTS, it means that another task with a smaller timestamp may arrive later. The task cannot be sent out, and the event processing is suspended.

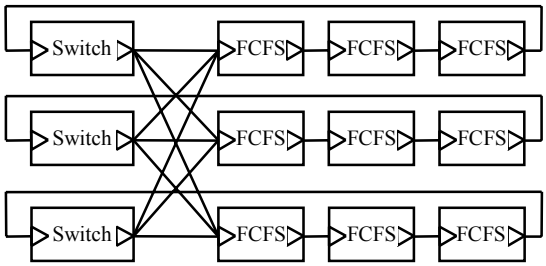


Figure 3. A CQN with 3 Switches and 9 FCFS Servers.

Bagrodia and Liao [21] proposed a technique to reduce the rollback distance for optimistic CQN simulation. They distinguished between the receive time of a task and the time at which a task can be served. When a straggler task arrives, the logical process needs to be rolled back only to the earliest time this message can be served. This approach bears some resemblance to the lookback-based protocols, for both algorithms are based on the observation that rollback to the timestamp of the straggler is unnecessary. However, lookback protocols can totally eliminate stragglers by deliberately delaying the execution of events that would raise lookback above the timestamp of the stragglers.

The lookback in the FCFS server requires no state-saving; the right position for a straggler task can be found by an insertion operation on an already sorted task list. However, if the buffer of the queue is finite and as a result may drop some tasks, modeling for lookback-based protocols is no longer trivial. Extra consideration must be taken in order to determine the right task to drop. In the traditional FCFS server, for instance, any task received after the queue size reaches the maximum limit must be discarded. This is not the case when a straggler arrives in a FCFS server that makes use of lookback. Indeed, another task in the queue with a receiving timestamp greater than that of the straggler may be removed from the queue. The search for such a task could be very difficult. In the simulation, each task is associated with an integer indicating its position in the queue if all events are timestamp ordered, and the dropping decision is made when a task is about to be serviced..

Such a lookback procedure with dropping control has some features of an optimistic simulation. The departure time of each task must be saved to enable calculating the position of stragglers. The departure times smaller than the LBTS can be discarded, which resembles fossil

collection. However, only the time, not the departure event, needs to be saved. The memory consumed is therefore much smaller than in the optimistic simulation.

We tested the performance of the two lookback-based protocols, LB-GVT and LB-EIT, on the CQN simulation. We assumed the queue capacity is infinite, and then added the dropping control algorithm described above (no tasks are actually dropped to avoid the effect of decreasing number of tasks). All experiments are conducted on a 500Mhz Quad Pentium III machine. Four CPUs are used in the parallel execution. Implicit heap queue is used to maintain the simulation event list.

Figure 4 shows the event processing rates of lookback-based protocols on the CQN with different configurations. Figure 5 shows the speedup of LB-GVT on 4 CPUs, compared with the sequential execution of LB-GVT. The size of the CQN is determined by two parameters: the number of servers in each tandem and the number of tandems (or the number of switches). Another parameter is the number of tasks initially in each FCFS server. We can see that LB-EIT performs poorly; it achieves some speedup only with greater density of tasks. This result is the same as those obtained for simulations with lookahead-based protocols, because relatively low activity requires large number of null messages to advance the simulation clock. We see also that the number of switches has a great impact on the performance of LB-EIT. With increased numbers of switches, the number of FCFS servers connected to a switch also increases, thus a switch must sent more null messages to the FCFS servers, making LB-EIT less efficient. We believe LB-EIT may outperform LB-GVT on systems with low connectivity.

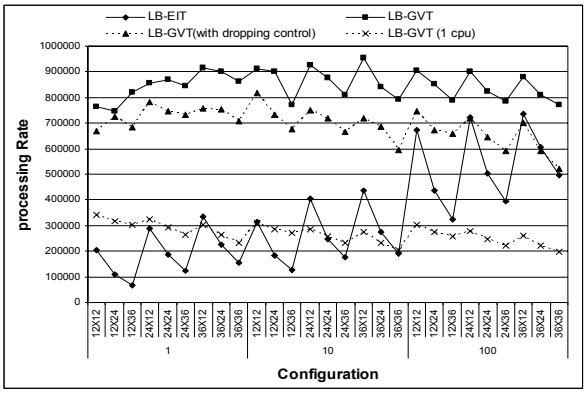


Figure 4. Performance of Lookback-Based Protocols in the CQN simulation.

The performance of LB-GVT with dropping control also drops when the task density is high. This is to be expected because as the queue and the departure time list become larger, the cost of maintaining the position for each task rises.

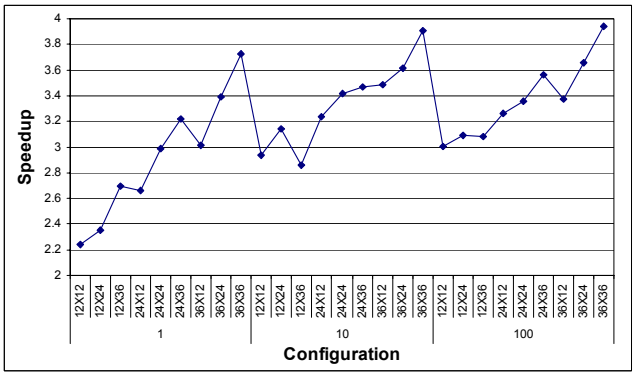


Figure 5: Speedup of LB-GVT on the CQN simulation using 4 CPUs

5. Conclusion

Lookback-based protocols introduce a new technique to exploit intra-logical process parallelism in simulation models. Surprisingly, the property of lookback, the ability to change the past, is closely related to the property of lookahead, the ability to predict the future. We have shown that in simulation models lookback is more common than lookahead. Furthermore, these two notions are complementary and can be exploited at the same time. It is of theoretical importance that lookback-based protocols allow conservative simulation to circumvent the critical path execution time limit.

However, the lookback-based protocols provide no general solution to PDES problems. Lookback does not exist in all simulation models, and even if it does exist, the lookback procedure may not be efficient. The utilization of lookback also complicates the simulation modeling, because out-of-timestamp execution has to be taken into consideration.

Despite of these two problems, we believe that our discovery of lookback points out a new direction for research in PDES. The feasibility and the performance of lookback-based protocols on a wide variety of simulation models have yet to be established by more extensive experiments. More efforts must be expanded to understand the semantics of the simulation time, which may lead to still more interesting approaches to PDES design.

Acknowledgment

This work was partially supported by the NSF Grant KDI-9873139. The content of this paper does not

necessarily reflect the position of policy of the U. S. Government – no official endorsement should be inferred or implied.

Bibliographies

[1] Fujimoto, R.M., *Parallel Discrete Event Simulation.* Communication of the ACM, 1990: p. 30-53.

[2] Bryant, R.E., *Simulation of Packet Communications Architecture Computer Systems*. 1977, Massachusetts Institute of Technology.

[3] Chandy, K.M. and J. Misra, *Distributed Simulation: A Case Study in Design and Verification of Distributed Programs.* IEEE Transactions on Software Engineering, 1979. **SE-5**: p. 440--452.

[4] Jefferson, D.R., *Virtual Time.* ACM Transactions on Programming Languages and Systems, 1985. **7**(3): p. 404-425.

[5] Jha, V. and R.L. Bagrodia, *Transparent Implementation of Conservative Algorithms in Parallel Simulation Languages*, in *Proceedings of the 1993 Winter Simulation Conference*. 1993. p. 677-686.

[6] Nicol, D. and X. Liu. *The Dark Side of Risk (What your mother never told you about Time Warp)*. in *Proceedings of the 1997 Workshop on Parallel and Distributed Simulation*. 1997. Austria. p. 188--195.

[7] Chen, G., B.K. Szymanski, and T. Caraco, *Multiparadigm Simulations in Modeling Spread of Lyme Disease*, in *2000 European Simulation Multi-Conference*. 2000.

[8] Sokol, L.M., *MTW: An Empirical Performance Study*, in *Proceedings of the 1991 Winter Simulation Conference*. 1991. p. 557-563.

[9] Wilsey, P.A., A.C. Palaniswamy, and S. Aji. *Rollback Relaxation: A Technique for reducing Rollback Costs in an Optimistically Synchronized Simulation*. in *International Conference on Simulation and Hardware Description Languages*. 1994. p. 143-148.

[10]Quaglia, F. and R. Baldoni, *Exploiting Intra-object Dependencies in Parallel Simulation.* Information Processing Letters, 1999. **70**(3): p. 119-125.

[11]Leong, H.V. and D. Agrawal, *Semantics-based Time Warp Protocols*. 1993, University of California, Santa Barbara.

[12]Dickens, P. and P. Reynolds. *SRADS with Local Rollback*. in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1990. p. 161--164.

[13]Reynolds, P., C. Weight, and J. Filder. *Comparative Analyses of Parallel Simulation Protocols*. in *Proceedings of the Conference on Winter Simulation Conference*. 1989. Washington. p. 671--679.

[14]Steinman, J. *Breathing Time Warp*. in *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*. 1993. San Diego. p. 109--118.

[15]Fujimoto, R. *Performance Measurements of Distributed Simulation Strategies*. in *Proceedings of the Distributed Simulation Conference*. 1988. p. 14--20.

[16]Lubachevsky, B.D., *Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks.* Communcation of the ACM, 1989. **32**(1): p. 111-123.

[17]Jefferson, D. and P. Reiher. *Supercritical Speedup*. in *Proceedings of the 24th Annual Simulation Symposium*. 1991. p. 159-168.

[18]Srinivasan, S. and P.F. Reynolds. *Super-Criticality Revisited*. in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*. 1995. Lake Placid, NY USA. p. 130--136.

[19]Gunter, M.A. *Understanding Supercritical Speedup*. in *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*. 1994. Edinburgh, Scotland. p. 81--87.

[20]Gafni, A. *Rollback Mechanisms for Optimistic Distributed Simulation*. in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1988.

[21]Bagrodia, R.L. and W.-T. Liao. *Transparent Optimizations of Overheads in Optimistic Simulations*. in *Proceedings of the 1992 Winter Simulation Conference*. 1992. p. 637-645.