J. King Saud Univ., Vol. 10, Comp. & Info. Sci., pp. 27-51 (A.H. 1418/1998)

Parallel and Distributed Simulation : Methodologies and Techniques

Hussam M. Soliman
Information Systems Department, College of Computer and Information Sciences
King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia

(Received 25 February 1996, accepted for publication 01 October 1996)

Abstract. An introduction to the field of Parallel and Distributed Simulation (PADS) is given. The capabilities and limitations of currently used PADS techniques are discussed. A review of the recently developed hybrid and adaptive PADS techniques is also given. Sample performance results of some PADS techniques are presented using a network of workstations.

1. Introduction

Parallel and distributed simulation is an area that has attracted considerable amount of interest in recent years. This interest arises from the fact that large simulations in many applications, such as extensive bank and organization simulations and model-validation simulations for decision support systems, consume enormous amounts of time on sequential computers [53]. In addition, the sequencing constraints on the execution of events are, in general, very complex and highly data-dependent.

In the process-based approach of PADS, the system is modelled as a set of interacting logical processes (LP's). Each logical process is a model of some component of the physical system, called a physical process. Interactions between logical processes in the model simulate interactions between physical processes in the real system. Each logical process contains a portion of the state corresponding to the physical process it models, as well as a local simulation clock. All interactions between physical processes are modelled by timestamped event messages sent between the corresponding logical processes. In order to avoid chronological errors, a logical process must process events in nondecreasing timestamp order [19].

Process-based parallel and distributed simulation algorithms fall broadly into two categories based on their synchronization scheme: conservative and optimistic. Conservative algorithms strictly avoid the possibility of incorrect sequencing of events by including strategies for determining which events are *safe* to process at each point in simulated time. As a result, this creates the potential for deadlock. Optimistic algorithms, on the other hand, use a detection and recovery approach which is based on detecting incorrect event execution sequences, and a *rollback* mechanism to recover.

The two main synchronization approaches to PADS, conservative and optimistic, have some limitations when the size and complexity of the simulation system increases. The conservative approach is limited by blocking overhead and sensitivity to *lookahead*. The optimistic approach is prone to cascading rollbacks and is limited by state saving overheads. Hybrid and adaptive approaches have been developed to resolve these limitations, while preserving potential advantages of each approach.

2. Conservative PADS Techniques

Conservative techniques are based on the idea of determining when it is safe to process an event. If a process contains an unprocessed event with timestamp T, and that process can determine that it is impossible for it to later receive another event with timestamp smaller than T, then the process can safely process that event. Processes containing no safe events must block which can lead to a deadlock situation, in general.

2.1 Deadlock-avoidance techniques

The first deadlock avoidance conservative PADS technique is the CMB algorithm developed by Chandy and Misra [9]. In this technique, it is required that an LP send messages over any outgoing link in nondecreasing timestamp order so that the last message received on an incoming link is a lower bound on the timestamp of any subsequent message that will be later received. A link clock is defined as the timestamp of the last received message, if the link buffer is empty, or as the timestamp of the message at the front of the link buffer, if the buffer is not empty. The link buffer is a FIFO queue in which arriving messages are stored in timestamp order. The process repeatedly selects the link with the smallest link clock as the next link from which to process messages. If the selected link has an empty buffer, the process is blocked. Deadlock situations arise when a cycle of empty queues with sufficiently small clock values exists because, in this case, each LP in the cycle must block. An example of a deadlock is shown in Fig. 1. In CMB, deadlock is avoided by using overhead *null messages*. A null message with timestamp t is sent from LP_i to LP_j to inform LP_j that no further messages would be sent on the link between them up to time t.

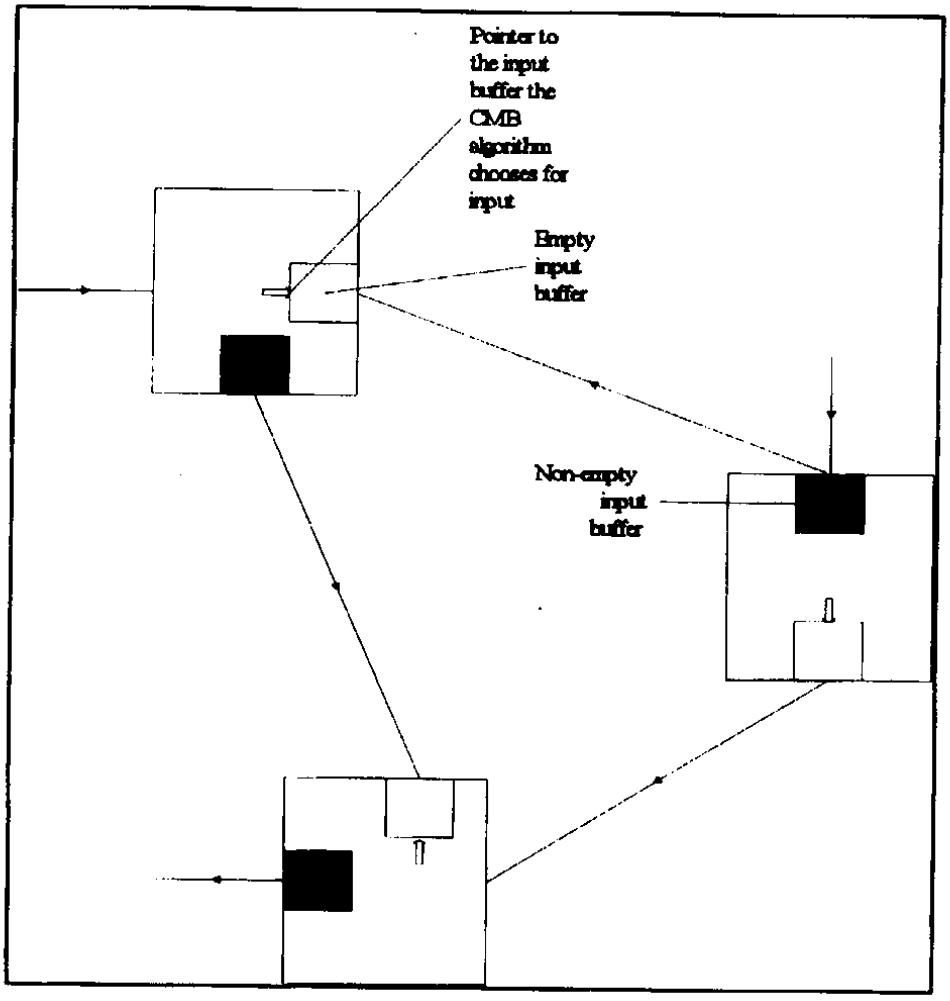

Fig. 1. A deadlock situation where every LP waits for input from an empty link buffer.

Several variations of the basic deadlock avoidance technique exist. A demand-driven null message approach was used by Nicol and Reynolds [40]. This approach helps reduce the amount of null message traffic. Wood and Turner [57] present another conservative algorithm in which null messages are responsible for both advancing simulation times and acquiring global knowledge about the system. The lookahead ability of the simulation is improved by analyzing the global knowledge carried by the protocol messages (referred to as the carrier-null messages). The algorithm is general enough to apply to simulations with arbitrary graphs.

2.2 Deadlock detection and recovery techniques

An alternative approach is to let the simulation scheme deadlock, detect the deadlock, and recover from it. A deadlock can be broken by observing that the message

containing the smallest timestamp is always safe to process. A deadlock detection and recovery scheme was presented by Chandy and Misra [10]. It is based on running the simulation until deadlock, and initiating a distributed computation whereby the various LP's can advance their local clocks. Another deadlock detection and recovery method, called the circulating marker, is described by Misra [37]. A marker is a special type of message that circulates among processes and declares a deadlock if it finds that the last n LP's that it visited were all *white* when it arrived at the LP it started from, where n is the number of links in the system. An LP is white if it has neither received nor sent a message since the last departure of the marker from that LP. The marker also carries the minimum of the next-event times for the white LP's it visits. When the marker detects deadlock, it knows the minimum next-event time and the LP at which this minimum occurs and, therefore, can restart this LP. Another technique, suggested by Liu and Tropper [29], detects specific types of cycles of blocked processes soon after their formation, and selects an LP which is suitable to break each kind of cycle. The algorithm distinguishes between three types of deadlock cycles: a deadlock cycle in which some LP's are waiting for messages while others are waiting for free memory buffers, a direct cycle of LP's each of which is trying to send a message to the full input buffer of its neighbor LP, and a direct cycle of LP's each of which is waiting for a message to appear in its empty input links. The algorithm first selects a leader which detects the deadlock and determines its category. It then finds the most appropriate LP to use in order to break the deadlock. Any blocked LP may initiate deadlock detection.

2.3 Other conservative techniques

Several other techniques to conservative PADS exist. One technique is to improve the *lookahead* ability of the logical processes. Lookahead is the time interval in the simulated future up to which an LP can predict with complete certainty all events it will generate. Lookahead is used in the deadlock avoidance approach to determine the timestamps assigned to null messages. A minimum timestamp increment of T translates into a lookahead of T because a process can guarantee that no new events will be generated with timestamp smaller than the present clock value plus T. Nicol [41] suggests to enhance the lookahead ability of a process by precomputing service times for jobs that have not yet arrived at that process in queuing network simulations. This precomputations cannot be applied, however, if the service time depends on parameters in the message.

The deadlock detection and recovery algorithm has been recognized, up to this point, as the only general conservative approach for simulating systems with no lookahead prediction. An alternative for systems with feedback loops and no lookahead prediction is presented by Lin and Lazowska [27]. It is suggested to reconfigure the system such that there is no feedback loop, and use the CMB algorithm without deadlock resolution to perform the simulation. Several other researchers have suggested exploiting properties of the network topology. De Vries [12] devises strategies to reduce the number

of null messages transmitted in the deadlock avoidance mechanism for specific applications that can be decomposed into feedforward and feedback networks.

Lubachevsky [31] uses a moving simulated time window to reduce the overhead associated with determining when it is safe to process an event. Qnly unprocessed events with timestamps larger than the lower edge of the window can be processed. Ayani and Rajaei [2] propose a conservative time window approach where they partition the physical system to be simulated into n disjoint subsystems, each of which is represented by an object for which a time window is identified. Events occurring in each window are independent of events in other windows and thus can be processed concurrently. The size of each window is calculated in each iteration of the algorithm using features of the system being simulated. Having windows with different sizes is shown to exploit more parallelism compared to the case where a global ceiling must be kept by all nodes.

Fujimoto [16] demonstrates that conservative algorithms can provide significant speedups over sequential event list implementations for some workloads containing moderate to high degrees of parallelism, even if there are many feedback loops in the logical process topology. He concludes that the lookahead ability of logical processes plays a critical role in determining the efficiency of the deadlock avoidance and deadlock detection and recovery algorithms. Loucks and Preiss [30] also examine the impact of exploiting lookahead on computation and communication overhead, and verify its important impact on performance. They conclude that between one half and two thirds of the computation load for a multiprocessor running a stochastic queuing network simulation are eliminated when user knowledge is added to the simulation.

2.4 Partioning algorithms

Performance studies have identified many issues which may affect the performance of conservative PADS. Among these are lookahead, network topologies, load balancing and process scheduling strategies [20]. The problem of static load balancing involves the partitioning of the simulation and mapping it onto a multicomputer network. The partitioning problem involves grouping nodes into blocks subject to two constraints: each block is approximately the same size, and the links between blocks are minimized. Objective functions are usually defined such that interprocessor communication conflicts are minimized, processor load is balanced, and the probability of sending a null message between processors is minimized. The partitioning problem has been identified to be computationally expensive. There are three major approaches to graph partitioning: graph theoretic , numerical, and min-cut heuristics. The heuristic approach allows the maximum flexibility in goal specification but yields suboptimal solutions.

Nandy and Loucks [38] presented a static partitioning algorithm for conservative parallel logic simulation. The algorithm attempts to minimize the communication

overhead and to uniformally distribute the execution load among the processors. It starts with an initial random partition and then iteratively moves processes between clusters until no improvements can be found. As a benchmark, they use the simulation of circuits modelled at the gate level on a message passing multicomputer composed of eight T-800 INMOS transputers. They report a 10-25% reduction in simulation time from the simulation time of a random partition.

Nandy and Loucks [39] later presented a parallel partitioning technique based on a min-cut iterative improvement algorithm which runs on the same multicomputer network running the simulation. Experiments were conducted to evaluate the algorithm performance for circuit simulation problems on transputers. The algorithm was modified to produce similar quality of final partition as the original sequential algorithm, and to increase the parallelism of the algorithm at the expense of quality. The algorithm relies on estimating the work load by pre-simulation runs.

The simulated annealing approach is a probabilistically directed iterative improvement scheme which is based on ideas from statistical mechanics and motivated by an analogy to the behavior of physical systems in the presence of a heat bath. This approach has the advantages that it is problem-dependent in the sense that substituting a few problem specific data structures and functions can make the algorithm be applied to many combinatorial optimization problems. It is also capable of handling multiple, potentially conflicting goals. Boukerche and Tropper [6] proposed the use of a simulated annealing algorithm with an adaptive search schedule to find good suboptimal partitions. They studied the performance of the algorithm for FCFS queuing network models on an IPSC/860 hypercube. Their results show a reduction of 25-35% of running time of the simulation compared to the results of a random partition.

3. Optimistic PADS Techniques

In optimistic approaches, we need not determine when it is safe to process an event; instead we determine when an error has occurred, and invoke a procedure to recover. This allows the simulator to exploit parallelism in cases where it is possible for causality errors to occur but they do not.

3.1 Time warp

An optimistic approach to PADS called Time Warp was proposed by Jefferson [23]. In Time Warp, a causality error is detected whenever an event message is received that contains a timestamp smaller than the local clock. The local virtual time (LVT) of a process is defined as the minimum receive time of all unprocessed messages. Processes can execute events and proceed in local time as long as they have inputs. Consequently, the local clock of a process can get ahead of its predecessors' local times. If the process

receives a message from a predecessor that has a timestamp smaller than the local clock, the process rolls back in simulated time and redoes its simulation to take into account the new message. This new message is called, in this case, a *straggler*. The process also sends *antimessages* to its successors to cancel the effects of previous erroneous messages. An antimessage annihilates the original message when they meet.

Assuming infinite memory, the Time Warp does not deadlock because individual processes do not deadlock as long as they have any inputs. The Time Warp approach has potentially greater speedup than conservative approaches at the expense of greater memory requirements. Moreover, in Time Warp, the topology of possible interactions between logical processes need not be fixed.

The performance of rollback mechanisms for the Time Warp algorithm has been studied analytically by Lin and Lazowska [26]. Time Warp with aggressive or lazy cancellation mechanisms, which will be explained later in this section, are shown to perform well for a process with a small simulated load intensity. Their results also indicate that message preemption has a significant effect on performance when the processor is highly utilized, the execution times of messages have high variance, and rollbacks occur frequently.

Substantial speedups have also been reported using the Time Warp protocol for simulating closed queuing networks [17] and for synthetic loads [20]. Using the direct cancellation strategy, he reports speedups as high as 57, using a 64 processor BBN Butterfly. The reported results indicate that Time Warp is capable of exploiting whatever parallelism is available in the simulation model without requiring extensive application-specific knowledge from the user. However, Loucks and Preiss [30] demonstrate that significant performance improvements can be obtained if one employs application specific knowledge to optimize execution. State-saving overhead can significantly degrade performance of Time Warp when the state size is increased [17].

Carothers, Fujimoto, and England [7] present empirical results on the performance of Time Warp in distributed computing environments. They conduct a series of Time Warp simulation experiments of a large granularity system and a low granularity system. They conclude that asynchronous message passing, where the sending process does not block until it receives acknowledgement that a message has been delivered, are much more suitable for distributed implementations. Moreover, the effectiveness of the Time Warp technique is significantly reduced for small granularity problems due to increasing communication delays. Large computation times for communication primitive execution are likely to affect Time Warp speedup than transmission latency. Effects of communication delay, in general, on efficiency (amount of rolled-back computation) is less significant than on speedup.

3.2 Variations of time warp

Numerous variations to the Time Warp mechanism have been proposed. Moving Time Windows [49] is another optimistic approach in which a time window in simulated time is defined so that its lower edge encompasses the unprocessed events containing the smallest timestamps. Events beyond the window must wait until the window has been advanced. The idea is to prevent incorrect computations from propagating too far ahead into the simulated future. A problem with this technique, however, is that time windows may block the progress of correct computations. Moreover, it is not clear how the size of the window should be determined.

Madisetti, Walrand and Messerschmitt [34] propose another optimistic mechanism called Wolf. In Wolf, a straggler message causes a process to send special control messages to other processes to stop the spread of the erroneous computation. One problem with this approach is that some correct computations may be unnecessarily frozen because the set of processors that might be affected by the erroneous computation may be much larger than the set that is actually affected.

A mechanism, called *direct cancellation* is proposed by Fujimoto [17] for shared memory multiprocessors. The technique depends on using pointers between events to locate those that need to be cancelled due to rollback. This eliminates the need for antimessages and provides an efficient mechanism for cancelling erroneous messages. Direct cancellation results in a distributed data structure that utilizes pointers to indicate which events can affect others. Cancellation of an incorrect computation due to processing some event amounts to a tree traversal.

3.3 Global virtual time

Global Virtual Time (GVT) is defined at real time t as the minimum of all local clocks at t and of the timestamps of all event messages that are in transit at t. Obtaining an accurate estimation of GVT is important for memory management and output commitment in optimistic simulation schemes. The process of reclaiming memory and committing irrevocable operations is referred to as *fossil collection*. GVT approximation is an example of a distributed monotonic computation, one which tries to find a good lower bound on the current value of some global infimum function f. For a cut C of the space-time diagram of the simulation, a global infimum function f(C) evaluated at C is defined as the infimum of all local values of virtual time taken at the moment of the cut at each LP and all timestamps of messages which cross the cut line (are in transit). A result proved by Mattern [35] states that f(C) is a lower bound on the value of the function f at the moment f(C) is determined, even if C is an inconsistent cut. This result represents the basis of GVT approximation algorithms which will be described in the next two subsections.

In general, any snapshot algorithm can be used to estimate GVT (for example, Chandy-Lamport snapshot algorithm [8]). However, specialized algorithms are preferred because they can be more efficient. Several algorithms have been proposed for computing GVT. In the following, GVT algorithms are divided into non-token-based algorithms and token-based algorithms.

3.3.1 Non-token-based GVT algorithms

One of the earliest GVT algorithms was proposed by Samadi [48]. The algorithms makes the assumption that all event messages are acknowledged. The idea behind the algorithm is to designate a real time moment *rt* at which to take a snapshot of the simulator (local virtual time and messages in transit). Since in reality it is not possible to determine an exact value for *rt*, we can set upper and lower bounds on *rt* at each process by creating intervals that overlap in time. Each LP forwards its information to a central GVT initiator process as it exits the interval. Let $[start_i, stop_i]$ denote an interval at the ith LP. The set of all such intervals must share a common real time *rt*. The algorithm runs in five phases:

1. The initiator broadcasts a GVT START message to all nodes. The receipt of the message at node *i* is $start_i$.
2. Each node responds to this message by sending an acknowledgement message to the initiator. The receipt time of the last acknowledgement message is real time *rt*.
3. The initiator broadcasts a GVT STOP message to all nodes. The receipt time of that message at node *i* is $stop_i$.
4. Node *i* responds by sending the initiator its local virtual time. After the initiator collects this message from all nodes, it computes GVT by minimizing all the LVT's.
5. The initiator broadcasts a new GVT update to all nodes.

Bellenot [5] proposed another GVT algorithm based on message routing graphs and a binary-tree-like forwarding technique to replace the costly broadcasts and collections of Samadi's algorithm. Collections can be done in the reverse order. This algorithm has run time O(1) on each node and O(logN) run time overall. However, there is an O(logN) one time message routing graph configuration cost on each node.

Lin and Lazowska [25] proposed an algorithm which is similar to Samadi's algorithm but which does not use an acknowledgment message for every single message. Rather, every message carries a sequence number and when a process gets a certain control message it sends to every neighboring process the smallest sequence number which is missing from that process. The neighboring process assumes that this message and all messages with larger sequence numbers are still in transit. Since a process keeps

the local minima of timestamps for unacknowledged messages as a function of sequence numbers, it is able to compute a lower bound on the timestamps.

Tomlinson and Garg [54] present a GVT algorithm which minimizes the latency time of the GVT calculations (time between its occurrence and detection). Unlike the previously described algorithms, this algorithm is designed for a reliable non-FIFO communication subsystem and does not require messages to be acknowledged which significantly reduces the message overhead of the simulation. The algorithm works by specifying a target virtual time TVT and an initiator to detect when GVT TVT.

D'Souza, Fan, and Wisely [14] present a GVT algorithm, called pGVT, which is capable of operating in environments which do not support FIFO message delivery and where message delivery failure may occur, and thus message acknowledgment is used. PGVT is a passive response GVT algorithm which operates with a.central GVT manager calculating new GVT values from information reported by LP's. Each LP monitors the advancement of GVT and reports new information only when failure to do so would inhibit GVT advancement. The main advantage of the algorithm is the elimination of one communication cycle to request GVT information from the LP's. This is done by monitoring the last reported GVT value and comparing it to the new value broadcast by the GVT manager. The GVT manager calculates new GVT values by minimizing overall local GVT values sent by the LP's. An LP is triggered to report a new local GVT value by the progression of GVT to the last reported local GVT value. This has the effect of reducing the load on the communication subsystem.

Bauer and Sporrer [4] propose an asynchronous GVT algorithm which also relies on passive observation of the LVTs and the contents of the communication channels. The LP's independently send information messages at arbitrary interval to a dedicated central process, which continuously determines GVT based on received information and in turn distributes the result periodically back to the LP's. The algorithm assumes reliable FIFO channels and uses an event numbering scheme similar to that used by Lin and Lazowska. The main advantage of the algorithm is that it requires, on the average, less than N messages (with N denoting the number of LP's) to calculate an updated GVT value.

3.3.2 Token-based GVT algorithms

Token-passing GVT algorithms have also been used. A token-based algorithm was proposed by Preiss [43] which imposes a ring topology for passing the token. The algorithm runs all phases of Samadi's algorithm together. The circulating token is forwarded from node i to $(i+1)\ mod\ N$ carrying the new GVT update from the first round, the local virtual time values for the second round, and the GVT start message for the third round. Varghese, Chamberlain, and Weihl [56] presented a marker-based GVT algorithm which is derived from Misra's circulating marker scheme for deadlock recovery [3]. This algorithm can be made to work over non-FIFO links, and its overhead

can be dynamically tuned based on computational load. The main idea is to pass a marker to circulate the network in a tour in such a way that it traverses each unidirectional link at least once. The marker starts the tour from a designated initiator and returns to it at the end of the tour. The marker has two fields: Current_Min and GVT. In addition, each logical process LP_i maintains a variable called $Local\_Min_i$ to store the lowest local clock value at LP_i since the last time the marker visited LP_i. $Local\_Min_i$ is updated only when the marker visits LP or at rollback. Current_Min stores the smallest $Local\_Min_i$ for every LP in the current tour. When the marker returns to the initiator, it uses Current_Min as the new GVT-estimate. The new GVT value is broadcast to other Logical processes by storing it in the GVT field.

General snapshot algorithms can also be used to approximate GVT. Chandy and Lamport [8] proposed a snapshot algorithm to capture a consistent global state for any distributed system. The algorithm assumes communication channels are reliable and FIFO, and uses markers as delimiters for the messages in transit in the channels. In this algorithm, a process which has not record its state and which receives a marker on channel c records its state, records the state of channel c as the empty sequence, and sends the marker along all outgoing channels on which a marker has not already been sent. Fig. 2 shows how a logical process broadcasts the marker to its neighbors with this channel flushing mechanism. If the receiving process has already recorded its state, it records the state of channel c as the sequence of messages received along c after the process recorded its state and before it received the marker along c. The algorithm can be initiated by a single process or several processes (in which case marker sequence numbers must be used). A simple way to collect all the recorded information is for each process to send the state information it recorded to the initiator. Non-FIFO-channel versions of this algorithm were proposed by Mattern [35].

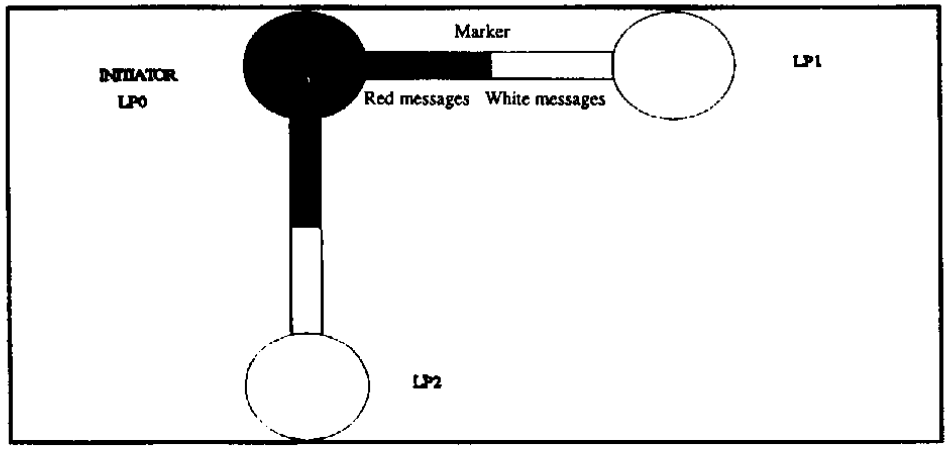

Fig. 2. Channel flushing in the chandy-lamport snapshot algorithm.

A slightly modified version of the Chandy-Lamport algorithm for FIFO links is proposed by Soliman and Elmaghraby [50]. As shown in Fig. 3, GVT-cycle initiation, in this version, is done in parallel by a GVT-manager process directly to each LP and not to one initiator LP. This saves the time the markers propagate in the network until all LPs are informed of the new cycle. The only overhead incurred by the modified algorithm would be the time needed to flush each outgoing channel with a marker, and the time needed to report the calculated local virtual time to the GVT manager. Reducing the duration of the GVT cycle results in more frequent fossil collection and, consequently, shorter message queues. This modification is shown to have a significant effect on the performance of the distributed simulator in terms of the overall speedup. Experimental performance comparisons with Samadi's GVT algorithm and Chandy-Lamport algorithm are also reported. A non-FIFO-channel algorithm is also presented by Soliman and Elmaghraby [50]. Steinman *et al.* [52] emphasize the importance of updating GVT efficiently in interactive distributed simulations and real-time applications. They also develop a GVT algorithm called SPEEDS GVT which also uses the idea of message flushing to account for messages in transit.

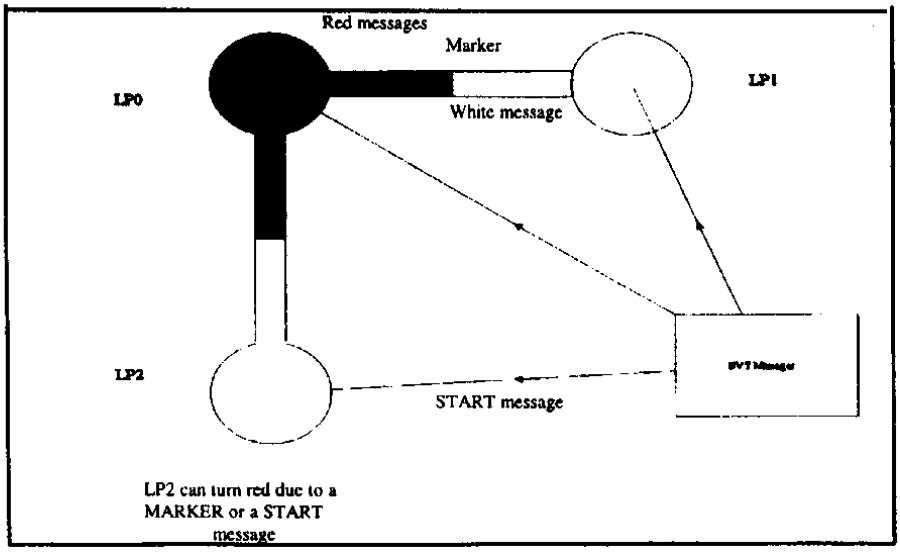

Fig. 3. The modified chandy-lamport snapshot algorithm .

3.4 Cancellation Strategies

In optimistic simulation systems, the simulation is divided into several objects which interact by sending event messages. Incorrect computations can arise because messages can be processed out of order and erroneous messages may be sent to other objects in a cascading fashion, as a result. Two methods are used for cancelling incorrect messages in Time Warp systems: *aggressive cancellation* and *lazy cancellation.* In aggressive cancellation, a process immediately sends antimessages when it rolls back. In

lazy cancellation, on the other hand, antimessages are not sent immediately after roll back. Instead, the process resumes execution forward in simulated time from its new clock value, and when it produces a message it compares it with the messages in its output queue. Only messages that are different from previously sent messages are transmitted, and only antimessages that are not reproduced in the forward computation are transmitted. The idea is to exploit the fact that not all of the work performed by an erroneous computation is incorrect. However, if the erroneous computation generated incorrect messages, lazy cancellation delays sending the corrective antimessages which allows incorrect computations to spread further than in the case of aggressive cancellation. In addition, lazy cancellation requires some additional overhead that is not incurred by aggressive cancellation. For example, it must determine if a matching antimessage already exists whenever a new message is generated. Also, it must check if any antimessages must be sent whenever the process's simulation clock is advanced.

Reiher *et al.* [45] present analytic results and performance measurements for both lazy and aggressive cancellation strategies and use applications that demonstrate the strengths of both techniques. They conclude that, depending on the application, aggressive or lazy cancellation can produce better performance although, for typical Time Warp applications, lazy cancellation shows slightly better performance. They recommend that the user be given the option of using lazy or aggressive cancellation.

A third cancellation strategy is *lazy re-evaluation*. If the state of the process is the same after processing a straggler event message as it was before and no new messages arrive, then the re-execution of rolled back events will be identical to the original execution. Therefore, there is no need to re-execute those events, and instead one can jump forward over these events. Lazy re-evaluation was implemented in the JPL Time Warp kernel, but later removed because it significantly complicated the Time Warp code [18].

3.5 Memory management and state saving

The Time Warp mechanism requires more memory to efficiently complete a parallel simulation than the equivalent sequential simulation. This is because Time Warp relies on rollbacks to synchronize the parallel computations which requires that the kernel store, in addition to pending events, some incorrect input messages that will later be cancelled, prior snapshots of each process's state, and an antimessage for each message that has been sent by the process.

A number of memory management schemes have been proposed to reduce the space usage of Time Warp. Some of these schemes reduce average space utilization, but cannot recover storage when the simulation runs out of memory. Other schemes are more adaptive and can run the simulation within the available memory and are able to recover memory on demand. Das and Fujimoto [11] efficiently implemented the cancelback

protocol on a shared memory multiprocessor implementation of Time Warp. They studied the performance of the simulator for three different workloads with different degrees of symmetry and concluded that severe asymmetry places severe demands on memory usage. They observed that, depending on the available memory and asymmetry in the workload, canceling back several events at one time may improve performance significantly. They also observed that a performance nearly equivalent to that with unlimited memory can be achieved with only a modest amount of memory depending on the degree of asymmetry in the workload.

In a Time Warp simulation, the state of each process must be saved regularly regardless of whether or not rollbacks occur. There are two approaches to reducing the overhead associated with state saving. Fujimoto *et. al.* [21] developed special-purpose hardware to support fast state saving. Another approach is to reduce the frequency of state savings. If a process with LVT equal to T_3 receives a straggler with timestamp $T_2 < T_3$ the process normally rolls back to T_2 and resumes execution from that point. However, if $T_1 < T_2$ is the timestamp of the last checkpointed event, the events with timestamps between T_1 and T_2 are reexecuted to produce the state of the process at T_2. This reexecution phase is called *coasting forward*. In this phase, any scheduling of future events is ignored because the purpose of this phase is to restore a process state that was not preserved. The selection of the checkpoint interval is a tradeoff between the total cost of checkpointing and the amount of re-execution in coast forward.

Lin *et al.* [28] proposed an optimization algorithm that quickly selects the optimal checkpoint interval during the execution of Time Warp simulation and takes into consideration the effect of the checkpoint interval on the rollback behavior. Performance of the algorithm is tested for stochastic closed queuing networks. The algorithm is found to find the true optimal checkpoint interval within a small number of iterations. Palaniswamy and Wilsey [42] analytically compare periodic state saving with a technique called incremental state saving. In incremental state saving, only the incremental changes in state are saved in the state queue as events are processed. They conclude that incremental state saving has the potential to outperform periodic state saving if the number of increments saved after each execution is small. Ronngren and Ayani [47] also analyze the effects of sparse checkpointing on the performance of Time Warp. They also present a method that allows each logical process to continuously adapt its state-saving interval based on its rollback behavior. They show that this method has low overhead and consumes less memory which improves performance. Fleischmann and Wilsey [15] analyze three techniques which dynamically or statically adjust the checkpointing interval including that of Ronngren and Ayani. They also propose a heuristic algorithm for this purpose. Their results show a significant difference in the performance of the implemented algorithms with the dynamic outperforming static algorithms.

4. Hybrid and Adaptive PADS Techniques

Conservative and optimistic approaches to PADS usually encounter limitations when the size and complexity of the simulated system increases. The blocking problem and the sensitivity to lookahead in the conservative protocols, and cascading rollbacks and state saving problems in the optimistic protocols are the key limitations for the respective approaches. Hybrid and/or adaptive approaches, with features of both Time Warp and conservative approaches, seem to provide the solution.

4.1 Adding optimism to conservative protocols

Reynolds [46] defines the *aggressiveness* of a PADS protocol as relaxing the requirement that messages be processed in a strict monotonic timestamp order. *Risk* is defined as passing messages that have been processed based on aggressive or inaccurate processing assumptions. Time Warp is an example of a maximal aggressiveness protocol with maximal risk.

To add optimism to a conservative protocol, the protocol may optimistically simulate the first event from its event list whenever a logical process is to be blocked. This is basically the approach followed by Dickens and Reynolds [13] and Rajaei, Ayani, and Thorelli [44]. This method, however, may not exploit potential parallelism since the results are kept local by each logical process until becoming secured.

4.2 Limiting optimism in optimistic protocols

Another method for integrating conservative and optimistic approaches is to limit the optimism of optimistic approaches. An optimistic scheme can be made less optimistic and approach a conservative one if the degree of risk is bounded to a certain limit. For example, Bounded Time Warp [55] is a technique that divides the simulation duration time into a number of equally sized intervals such that all events inside the current interval are started before the next interval is started. In Wolf [34], whenever a rollback occurs, special messages are broadcast in order to stop the spread of the erroneous computation. In Moving Time Windows [49] a window boundary is used to limit the optimistic computation using a window with a predefined size.

Madisetti, Hardaker and Fujimoto [33] propose the MIMDIX system which has additional processes called Genie processes which are used at regular intervals to probabilistically determine whether to resynchronize the logical processes in order to slow down their progress. McAffer [36] uses a sliding window mechanism for both sending and receiving messages which could lead to deadlock situations.

4.3 Switching between optimism and conservatism

A third method for integrating conservative and optimistic approaches is to switch between optimism and conservatism. It would be desirable, in this case, to determine the

degree of aggressiveness and risk dynamically. This requires extra information to be stored and processed which can degrade performance. In Composite ELSA [1], a node can switch between conservative and optimistic modes by providing extra information which is used to determine if an event is certain or guessed. In Adaptive Time Warp [3], a tuneable blocking window is defined such that a processor is blocked for a window duration whenever it experiences an abnormally high number of rollbacks.

Jha and Bagrodia [24] propose a methodology to unify conservative and optimistic techniques in an attempt to create an adaptable protocol which allows conservative and optimistic processes to coexist. This has the advantage of allowing different subsystems with contradictory characteristics to choose to be conservative or optimistic depending on their runtime behavior. Furthermore, logical processes whose behavior changes dynamically would be allowed to switch at runtime to the opposite mode of operation. It is not clear, however, which criteria should be used to decide when to switch modes.

Hamnes and Tripathi [22] also address the adaptivity issue by proposing a method which selects the optimum operating point for each logical process in the continuum of protocols from pure conservative to pure optimistic. This is done by defining a blocking window for individual channels for each process whose size is regularly optimized based on statistics gathered by the process at runtime.

4.4 The aggressive adaptive-risk approach

Protocols that add optimism to the conservative approach usually cannot support dynamic configurations which is desirable in large and complex simulations. The main drawback, of protocols which limit optimism is to determine an appropriate limiting boundary to significantly improve performance for large scale simulations. Switching between conservative and optimistic modes of operation is attractive for systems where the behavior of the various components change dynamically. However, the determination of the needed extra information and how it is to be used to arrive at a decision as to which mode of operation is more appropriate, is not clear.

A novel scheme was developed by Soliman and Elmaghraby [51] for the unification of the conservative and optimistic paradigms in a clustered and adaptive manner. The idea is based on grouping processes with similar characteristics into ensembles or clusters. Each cluster internally runs a modified version of the Time Warp protocol, while interacting with other clusters. The internal protocol allows the member processes the maximum degree of aggressiveness as in the original Time Warp protocol. However, the internal protocol requires the use of special buffers for cluster output messages in order to control the degree of risk. Fig. 4 shows the architecture of a distributed simulator using AAR. By holding cluster output messages in buffers, the protocol can effectively control cascading rollbacks, a problem associated with Time

Warp, since a large proportion of straggler messages could be generated before the release of the buffered messages to other clusters. Therefore, by controlling the duration of the real-time period in which the messages are stored in the cluster buffers, and the range of message timestamp values that will be released each time, one can adaptively control the degree of risk in order to be able to set a bound on the probability of an inter-cluster rollback, and maximize the rate of change of simulated time with respect to real time for each link.

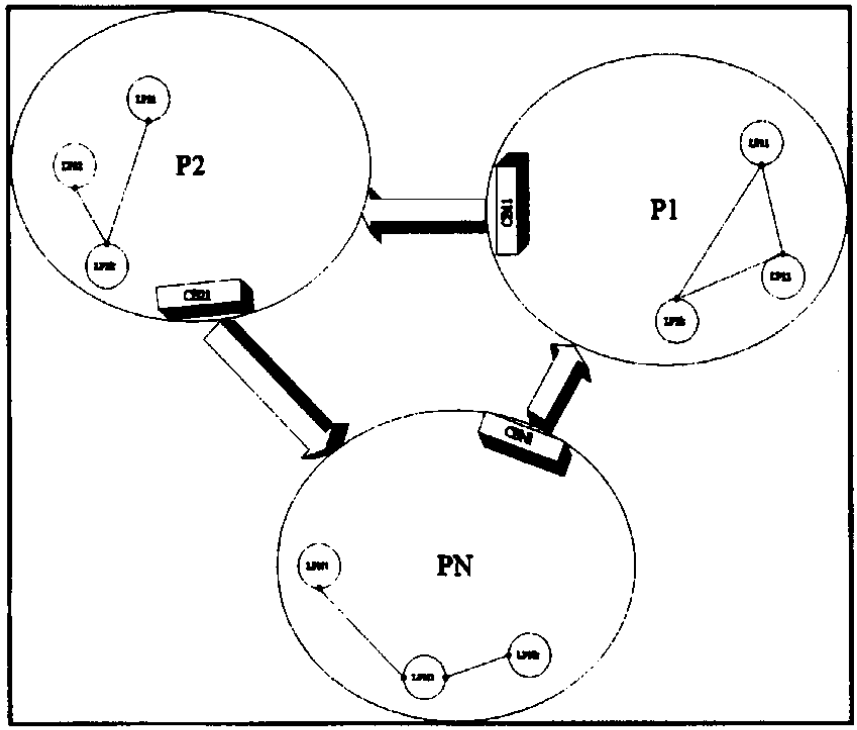

Fig. 4. The architecture for the aggressive adaptive-risk scheme.

5. Experimental Performance Results

Experiments are conducted to compare the performance of the CMB and the AAR techniques with that of Time Warp using a network of workstations. Description of these experiments and their results are presented in this section.

5.1 Description of the experiments

Implementations of the CMB and Time Warp protocols are developed on a network of four HP9000/735 workstations interconnected by a 10Mbit/sec Ethernet. Each machine runs 4 LP's, and the 16 LP's are connected as a 4x4 torus topology as

shown in Fig. 5. A synthetic simulation workload is used for the experiments. The distribution of the computation time per event and the routing probability are parameters of the workload. The torus topology is divided into 4 cells of 4 LP's each and each cell is mapped to one processor.

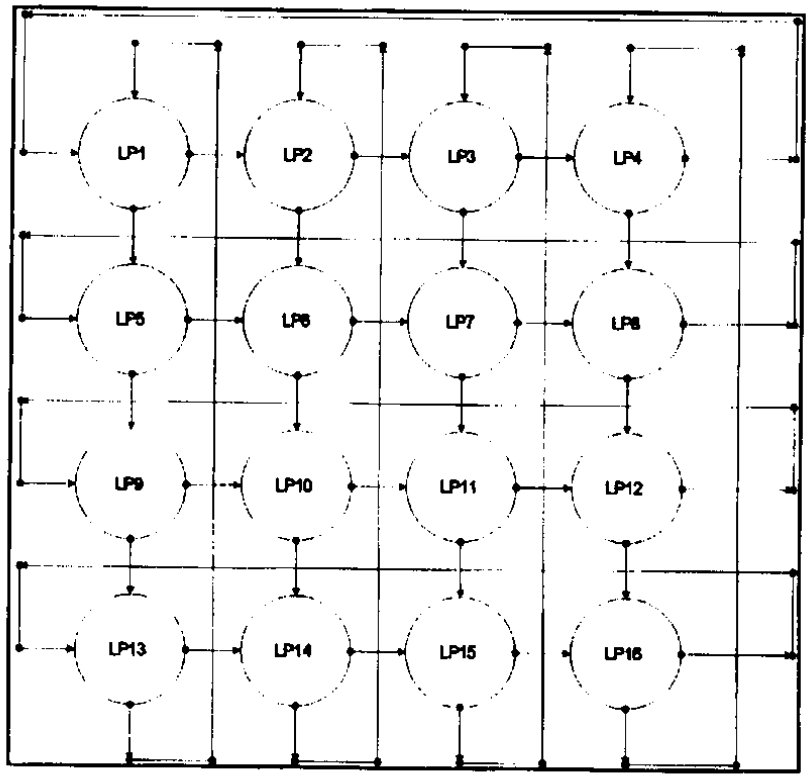

Fig. 5. The 4x4 torus topology.

5.2 Results

The performance of the CMB and Time Warp protocols was studied on the above testbed. It is observed in Fig. 6.a that speedup increases for CMB as the event computational time delay is increased from 10 ms to 50 ms. This verifies that a large computation granularity is required for a parallel simulator to yield acceptable performance. Since the message population determines the amount of parallel activity that can occur in the simulation, increasing the message population from 8/LP to 32/LP is seen to increase speedup. Fig. 6.b illustrates the effect of lookahead on speedup in CMB. The *lookahead ratio* (LAR) is defined as the mean service time an LP provides for each event to the minimum service time [16]. A low value of LAR means better lookahead properties. It is seen that a high value of LAR=5 results in speedups below one, whereas a low value of 1.5 results in a reasonable 1.5 for a message population of 8/LP. This clearly shows the sensitivity of CMB performance to the simulation lookahead

properties. In fact, it is required in CMB that each feedback loop in the simulated system topology contain at least one LP with nonzero lookahead value.

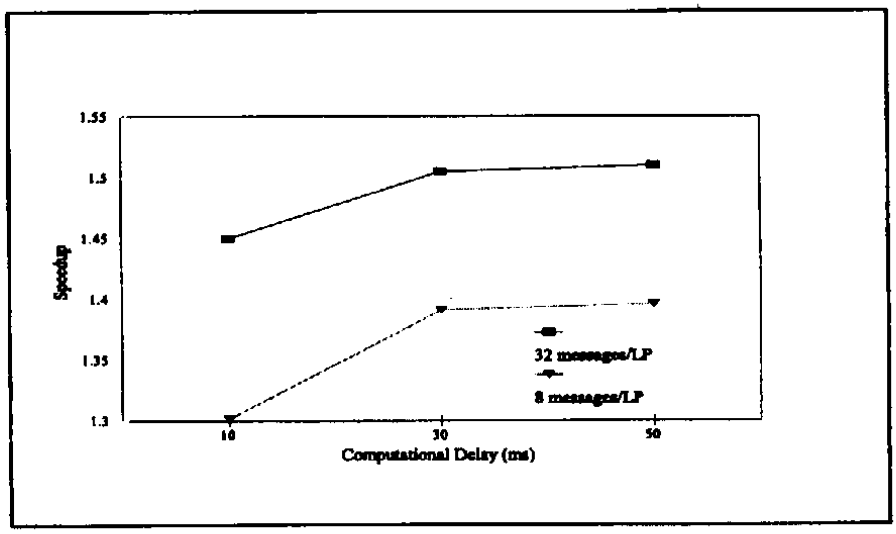

Fig. 6 a. CMB speedup vs. computational delay for LAR = 1. 5.

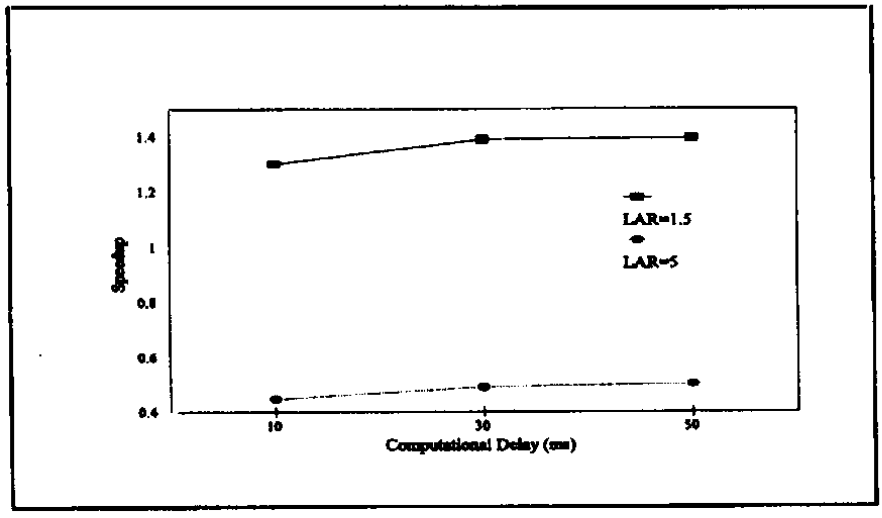

Fig. 6 b. CMB speedup vs. computational delay for message population 8/LP.

The performance of the Time Warp protocol is also studied on the above testbed. The modified version of the Chandy-Lamport snapshot algorithm was used for GVT computation. From Fig.7 it is clear that Time Warp outperforms CMB for this simulation. This is due to the optimistic nature of Time Warp which, unlike CMB, allows

it to utilize whatever parallelism is present in the system. Moreover, it is seen that increasing event computational granularity also increases the obtainable Time Warp speedup. This result is accounted for by the low network bandwidth requirements for large event granularity simulations. Increasing the message population to as high as 50/LP is seen to cause performance degradation. This is due to the increase in the time required for event list insertions and deletions, a problem not present in conservative algorithms since they use FIFO queues.

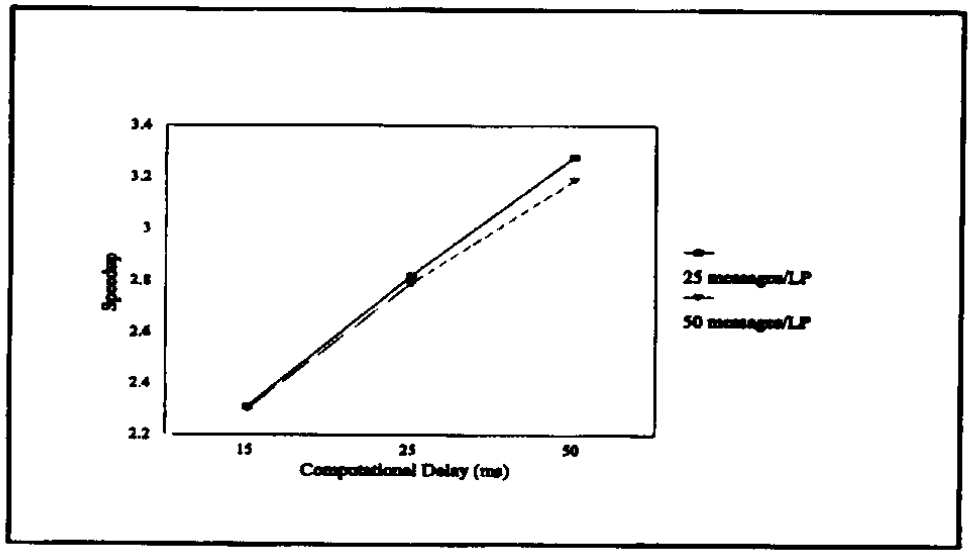

Fig. 7. Time warp speedup vs. computational delay for message populations 25/LP and 50/LP.

Figure 8 shows the speedup plot for the Clustered Parallel Simulator (CPS), a nonadaptive implementation of the aggressive adaptive-risk technique (AAR) discussed above, and that of Time Warp versus the simulated-time window size associated with the AAR technique. The CPS implementation is also developed on a network of eight HP 9000/735 workstations interconnected by a 10Mbit/sec Ethernet. Each of the eight machines runs 16 logical processes connected as a 4x4 torus forming an overall 8x16 torus topology. A fixed initial message population of 100 messages per logical process is used. It is also assumed that the state-vector size is zero and the load is uniform. Fixed simulated-time window sizes of .1, .3, .5 and 1 simulated-time units are selected after a few initial runs. Three cases for the CPS real-time release intervals are considered: release every execution of the main program loop, release every 5th execution of the main program loop, and release every 10th execution of the main program loop.

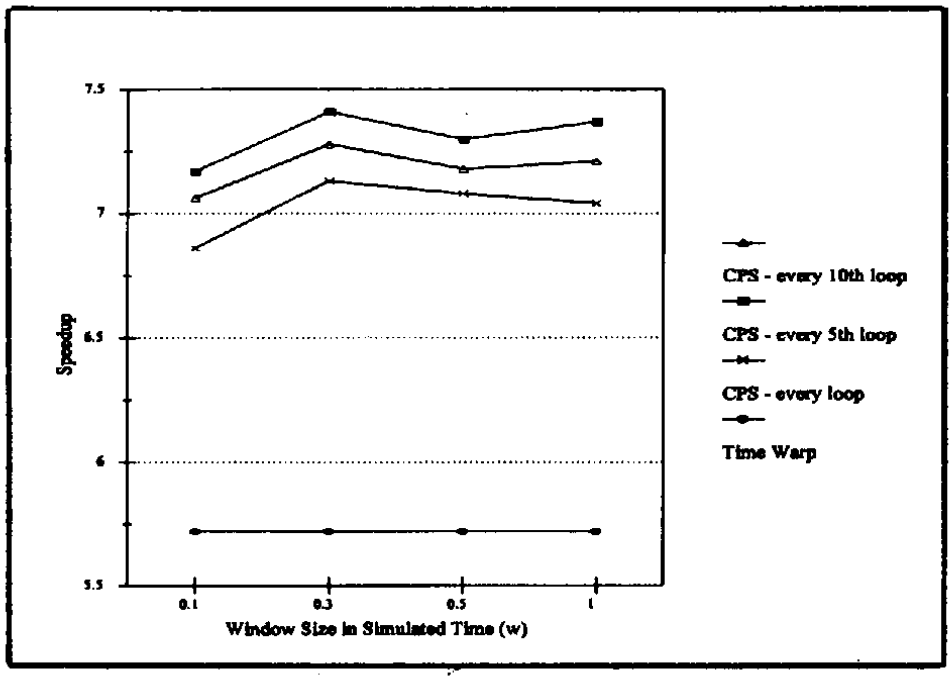

Fig. 8. Speedup performance of CPS compared to that of time warp.

The performance results of Fig. 8 show that a speedup of about 5.7 was achieved by the Time Warp simulator using the 8-machine testbed. A noticeable speedup improvement of about 1.5 is observed for the three cases of CPS indicating that the output buffering mechanism used in the AAR scheme was successful in limiting the overheads associated with processing external stragglers. This decrease in overhead is attributed to a decrease in the number of rollbacks in the destination cluster and local message cancellation in the cluster buffer of the source cluster. It is also observed that the effect of varying the real-time message release interval was almost negligible. Moreover, varying the (fixed) simulated-time window size also seems to have a small impact on speedup, although a general trend of increase with increasing window size is observed.

6. Conclusion

An extensive review of the state of the art in parallel and distributed simulation methodologies has been presented. Emphasis has been on recent research directions towards developing hybrid and adaptive techniques. Hopefully, these techniques will overcome the shortcomings of existing PADS techniques. Sample performance data, using a network of workstations, are reported to demonstrate some of the capabilities and limitations of both conservative and optimistic techniques.

References

[1] Arvind, D. K. and Smart, C. R. "Hierarchical Parallel Discrete Event Simulation in Composite ELSA." *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, (1992), 147-156.

[2] Ayani, R. and Rajaei, H. "Parallel Simulation Using Conservative Time Windows." *Proceedings of the Winter Simulation Conference*, (1992), 709-717.

[3] Ball, D. and Hoyt, S. "The Adaptive Time Warp Concurrency Control Algorithm." *Proceeding of the SCS Multiconference on Distributed Simulation*, (1990), 174-177.

[4] Bauer H. and Sporrer, H. "Distributed Logic Simulation and an Approach to Asynchronous GVT Calculation."*Proceedings of the 6th Workshop on Parallel and Distributed Simulation*,(1992), 205-208.

[5] Bellenot, S. "Global Virtual Time Algorithms." *Proceeding of the SCS Multiconference on Distributed Simulation*, (1990), 122-127.

[6] Boukerche, A. and Tropper, C. "A Static Partitioning and Mapping Algorithm for Conservative Parallel Simulations."*Proceedings of the 8th Workshop on Parallel and Distributed Simulation*,(1994),164-172.

[7] Carothers, C. D., Fujimoto, R. M. and England, P. "Effect of Communication Overheads on Time Warp Performance: An Experimental Study." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, (1994), 118-125.

[8] Chandy K. M. and Lamport, L. "Distributed Snapshots: Determining Global States of Distributed Systems." *ACM Transactions on Computer Systems*, 3 (1985), 63-75.

[9] Chandy, K. M. and Misra, J. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." *IEEE Transactions on Software Engineering*, 5, No. 5 (1979), 440-452.

[10] Chandy, K. M. and Misra, J. "Asynchronous Distributed Simulation via a Sequence of Parallel Computations." *Communications of the ACM*, 24, No. 11 (1981), 198-206.

[11] Das, S. R. and Fujimoto, R. M. "A Performance Study of the Cancelback Protocol for Time Warp." *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, (1993), 135-142.

[12] De Vries, R. C. "Reducing Null Messages in Misra's Distributed Discrete Event Simulation Method." *IEEE Transactions on Software Engineering*, 16, No. 1 (1990), 82-91.

[13] Dickens, P. M. and Reynolds, P. F. "SRADS with Local Rollback." *Proceeding of the SCS Multiconference on Distributed Simulation* (1990), 161-164.

[14] D'Souza, L., Fan, X. and Wilsey, P. "pGVT: An Algorithm for Accurate GVT Estimation." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, (1994), 102-109.

[15] Fleischmann, J. and Wilsey, P. A."Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators."*Proceedings of the 9th Workshop on Parallel and Distributed Simulation*,(1995), 50-58.

[16] Fujimoto, R. M. "Performance Measurements of Distributed Simulation Strategies." *Transactions of the SCS*, 6, No. 2 (1989), 89-132.

[17] Fujimoto, R. M. "Time Warp on a Shared Memory Multiprocessor." *Transactions of the SCS*, 6, No. 3 (1989), 211-239.

[18] Fujimoto, R. M. "Optimistic Approaches to Parallel Discrete Event Simulation." *Transactions of the SCS*, 7, No. 2 (1990), 153-191.

[19] Fujimoto, R. M. "Parallel Discrete-Event Simulation." *Communications of the ACM*, 33, No. 10 (1990), 31-53.

[20] Fujimoto, R. M. "Performance of Time Warp Under Synthetic Workloads." *Proceeding of the SCS Multiconference on Distributed Simulation*, (1990), 23-28.

[21] Fujimoto, R. M., Tsai, J. and Gopalakrishnan, G. "Design and Evaluation of the Rollback Chip: Special Purpose Hardware for Time Warp."*IEEE Transactions on Computers*, 41, No. 1 (1992),68-82.

[22] Hamnes, D. O, and Tripathi, A. "Investigations in Adaptive Distributed Simulation." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, (1994), 20-23.

[23] Jefferson, D. R. "Virtual Time." *ACM Transactions on Programming Languages and Systems*, 7, No. 3 (1985), 404-425.

[24] Jha, V. and Bagrodia, R. L. "A Unified Framework for Conservative and Optimistic Distributed Simulation." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, (1994), 12-19.

[25] Lin, Y. B. and Lazowska, E. D. "Determining the Global Virtual Time in a Distributed Simulation." *Proceedings of the International Conference on Parallel Processing*, (1990), 201-209.

[26] Lin, Y. B. and Lazowska, D. L. "A Study of Time Warp Rollback Mechanisms." *ACM Transactions on Modeling and Computer Simulation*, 1, No. 1 (1991), 51-72.

[27] Lin, Y. B. and Lazowska, E. D. "Reducing the Overheads of Conservative Parallel Simulation Without Lookahead." *International Journal in Computer Simulation*, 3 (1993), 231-259.

[28] Lin, Y. B., Preiss, B. R., Loucks, W. M. and Lazowska, E. D. "Selecting the Checkpoint Interval in Time Warp Simulation."*Proc. of the 7th Workshop on Parallel and Distributed Simulation*, (1993), 3-10.

[29] Liu, L. Z. and Tropper, C. "Local Deadlock Detection in Distributed Simulations." *Proceeding of the SCS Multiconference on Distributed Simulation*, (1990), 64-69.

[30] Loucks, W. M. and Preiss, B. R. "The Role of Knowledge in Distributed Simulation." *Proceedings of the SCS Multiconference on Distributed Simulation*, (1990), 9-16.

[31] Lubachevsky, B. D. "Efficient Distributed Event-Driven Simulations of Multiple-loop Networks." *Communications of the ACM*, 32 (January 1989), 111-123.

[32] Lubachevsky, B. D., Shwartz, A. and Weiss, A. "An Analysis of Rollback-Based Simulation." *ACM Transactions on Modeling and Computer Simulations*, 1, No. 2 (1991), 154-193.

[33] Madisetti, V., Hardaker, D. and Fujimoto, R. "The MIMDIX Operating System for Parallel Simulation." *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, (1992), 65-74.

[34] Madisetti, V., Walrand, J. and Messerschmitt, D. "WOLF: A Rollback Algorithm for Optimistic Distributed Simulation Systems." *Proceedings of the Winter Simulation Conference*, (1988), 296-305.

[35] Mattern, F. "Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation." *Journal of Parallel and Distributed Computing*, 18 (1993), 423-434.

[36] McAffer, J. "A Unified Distributed Simulation System." *Proceedings of the Winter Simulation Conference*, (1990), 415-422.

[37] Misra, J. "Distributed Discrete-Event Simulation."*ACM Computing Surveys*,18, No.1(March 1986), 39-55.

[38] Nandy, B. and Loucks, W. M. "An Algorithm for Partitioning and Mapping Conservative Parallel Simulation onto Multicomputers." *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, (1992), 139-146.

[39] Nandy, B. and Loucks, W. M. "On a Parallel Partitioning Technique for Use with Conservative Parallel Simulation." *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, (1993), 43-51.

[40] Nicol, D. M. and Reynolds, P. F., Jr. "Problem Oriented Protocol Design." *Proceedings of the Winter Simulation Conference*, (1984), 471-474.

[41] Nicol, D. M. "Parallel Discrete-Event Simulation of FCFS Stochastic Queuing Networks." *SIGPLAN Not.*, 23, No. 9 (1988), 124-137.

[42] Palaniswamy, A. C. and Wilsey, P. A. "An Analytical Comparison of Periodic Checkpointing and Incremental State Saving." *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, (1993), 127-134.

[43] Preiss, B. "The Yaddes Distributed Discrete Event Simulation Specification Language and Execution Environments." *Proceedings of the SCS Winter Multiconference*, (1989), 139-144.

[44] Rajaei, H., Ayani, R. and Thorelli, L. E. "The Local Time Warp Approach to Parallel Simulation." *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, (1993), 119-126.

[45] Reiher, P. L., Fujimoto, R. M., Bellenot, S. and Jefferson, D. R. "Cancellation Strategies in Optimistic Execution Systems." *Proc. of the SCS Multiconference on Distributed Simulation*, (1990), 112-121.

[46] Reynolds, P. "A Spectrum of Options for Parallel Simulation." *Proceedings of the Winter Simulation Conference*, (1988), 325-332.

[47] Ronngren, R., Rajaei, H., Popescu, A., Liljenstam, M., Ismailov, Y. and Ayani, R. " Parallel Simulation of a High Speed LAN." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, (1994), 132-138.

[48] Samadi, B. *Distributed Simulation Algorithms and Performance Analysis*, Ph.D. Dissertation. Los Angeles, CA, USA: University of California, 1985.

[49] Sokol, L. M. and Stucky, B. K. "MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm." *Proceeding of the SCS Multiconference on Distributed Simulation*, (1990), 169-173.

[50] Soliman, H. M. and Elmaghraby, A. S. "An Improved Chandy-Lamport Snapshot Algorithm for GVT Approximation in Distributed Simulations." *Proceedings of the 8th International Conference on Parallel and Distributed Computing Systems*, (1995), 473-477.

[51] Soliman, H. M. and Elmaghraby, A. S. "An Efficient Clustered Adaptive-Risk Technique for Distributed Simulation." *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, (1996), 383-391.

[52] Steinman, J. S., Lee, C. A., Wilson, L. F. and Nicol, D. M. "Global Virtual Time and Distributed Synchronization." *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, (1995), 139-148.

[53] Strezova, Z. "A Procedure for Decision Support Systems Design: Modelling and Simulation Environment." In A. Sydow, S. G. Tzafestas and R. Vichnevetsky (Eds.): *System Analysis and Simulation I: Theory and Foundations*. Berlin: Springer-Verlag, 1988.

[54] Tomlinson, A. I. and Garg, V. K. "An Algorithm for Minimally Latent Global Virtual Time." *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, (1993), 35-42.

[55] Turner, S. J. and Xu, M. Q. "Performance Evaluation of the Bounded Time Warp Algorithm." *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, (1992), 117-126.

[56] Varghese, G., Chamberlain, R. and Weihl, W. "The Pessimism behind Optimistic Simulation." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, (1994), 126-131.

[57] Wood, K. R. and Turner, S. J. "A Generalized Carrier-Null Method for Conservative Parallel Simulation." *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, (1994), 50-57.

المحاكاة المتوازية و الموزّعة : أساليب وطرق

حسام سليمان

قسم نظم المعلومات، كلية علوم الحاسب والمعلومات، جامعة الملك سعود،

ص ب ٥١١٧٨، الرياض ١١٥٤٣، المملكة العربية السعودية

(قدّم للنشر في ١٩٩٦/٢/٢٥م؛ وقبل للنشر في ١٩٩٦/١٠/١م)

ملخص البحث. تعرض هذه المقالة مقدمة في مجال الأساليب المستخدمة حاليا للمحاكاة المتوازية والموزّعة وكذلك محدودية هذه الأساليب. كما تستعرض هذه المقالة الأساليب المهجّنة والمتكيّفة والتي تمّ تطويرها حديثا. وأخيرا تقدم المقالة عيّنة من نتائج أداء بعض الطرق المحاكاة المتوازية والموزعة. وتناقش المقالة قدرات الأساليب المختارة للمحاكاة الموزّعة والتي تعمل علي شبكة من محطات العمل .