Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Synchronization methods in parallel and distributed discrete-event simulation

Shafagh Jafer [a,*], Qi Liu [b], Gabriel Wainer [c]

[a] Milwaukee School of Engineering, 1025 N. Broadway, Milwaukee, WI 53202-3109, USA
[b] IBM, T.J. Watson Research Center, 1101 Kitchawan Road, Route 134, Yorktown Heights, NY 10598, USA
[c] Department of Systems and Computer Engineering, Carleton University, 1125 Collonel By Drive, Ottawa, ON, Canada K1S 5B6

ARTICLE INFO

Article history:
Received 27 September 2011
Received in revised form 6 August 2012
Accepted 8 August 2012
Available online 16 October 2012

Keywords:
Discrete-event simulation
Conservative approach
Optimistic mechanism

ABSTRACT

This work attempts to provide insight into the problem of executing discrete event simulation in a distributed fashion. The article serves as the state of the art in Parallel Discrete-Event Simulation (PDES) by surveying existing algorithms and analyzing the merits and drawbacks of various techniques. We discuss the main characteristics of existing synchronization methods for parallel and distributed discrete event simulation. The two major categories of synchronization protocols, namely conservative and optimistic, are introduced and various approaches within each category are presented. We also present the latest efforts towards PDES on emerging platforms such as heterogeneous multicore processors, Web services, as well as Grid and Cloud environment.

1. Parallel discrete-event simulation

With the computing power and advanced software available today, modeling and simulation has become a cost-effective tool for detailed analysis of a broad array of natural and artificial systems. Parallel and distributed simulation is widely accepted as the technology of choice to speed up large-scale discrete-event simulation and to promote reusability and interoperability of simulation components. Parallel Discrete-Event Simulation (PDES) has received increasing interest as simulations become more time consuming and geographically distributed. A rich literature has already been developed in the last three decades, taking advantage of the increasing availability of highly parallel and distributed computing platforms. It has been several years since the last survey by Perumalla [5] and a refreshed review of the latest developments would be needed for us to ponder new research initiatives, especially on emerging platforms such as many-core processors, Internet-scale simulation environments, and cloud-based virtualized infrastructures.

In parallel and distributed simulations, the entire simulation task is divided into a set of smaller subtasks with each executed on a different processor or node. Hence, the simulation system is viewed as a collection of concurrent processes, each modeling a different part of the physical system and executing on a dedicated processor in a sequential fashion. These processes communicate with each other by exchanging time-stamped event messages. An event refers to an update to simulation system state at a specific simulation time instant. Throughout the simulation, events arrive at destination processes, and depending on the delivery ordering system of the simulation, they are processed differently. The two commonly used orderings are (1) receive-order and (2) timestamp-order. With the first type, events are delivered to the destination processes when they arrive at the destination. On the other hand, with the timestamp-order, events are delivered in non-decreasing order of their timestamp, requiring runtime checks and buffering to ensure such ordering.

* Corresponding author. Tel.: +1 276 439 9051; fax: +1 613 520 5727.
E-mail address: jafer@msoe.edu (S. Jafer).

1569-190X/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2012.08.003

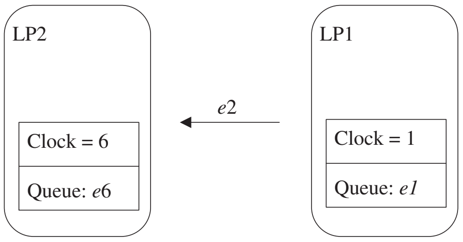

Fig. 1. Causality error scenario.

In discrete-event simulation, the operation of a system is represented as an ordered sequence of events, where each event occurs at an instant in time. A Parallel Discrete-Event Simulation (PDES) consists of Logical Processes (LPs) acting as the simulation entities, which do *not* share any state variables, and interact with each other merely through exchanging time-stamped event messages [1]. Each LP maintains a clock to advance its simulation time, coordinate the events' delivery, and assign a timestamp to those events it sends out to other LPs. In general, each LP is mapped to a physical processor of a parallel computing system, but if the number of LPs exceeds the number of available processors, multiple LPs are mapped to a single physical processor. The LPs that are allocated on the same processor typically maintain a single Future Event List (FEL) to schedule event execution in a *sequential* manner. The major challenge in PDES is being able to produce exactly the same results as in a sequential execution of the simulation program. This requires a precise and accurate synchronization of all the LPs in the system since data and computation distribution may result in different errors related to the concurrent processing of the simulation messages. Synchronization among these LPs is violated when one of the LPs receives an out of order event. An event is said to be out of order, if it is received in the past (i.e. the arrival time of the event is older than the current clock time of the recipient LP). Such violation is referred to as *causality error*. Fig. 1 exemplifies a causality error where two LPs, each with one event in its input queue, process their events simultaneously. When LP1 executes event *e1* (whose timestamp is 1), it generates and sends a new event message *e2* to LP2 (with a timestamp of 2). However, at this time, LP2 has already processed *e6* and advanced its local clock to 6. Consequently, the arrival of *e2* at LP2 violates causality, and an error occurs.

To ensure correct synchronization, a PDES system must satisfy the following necessary and sufficient condition, termed as the Local Causality Constraint by Fujimoto [2].

<u>Local Causality Constraint</u>*: A discrete-event simulation, consisting of LPs that interact exclusively by exchanging timestamped messages obeys the local causality constraint if and only if each LP processes events in non-decreasing timestamp order.*

To satisfy the local causality constraint, different synchronization techniques have been proposed for PDES systems which generally fall into two major classes of synchronization: *conservative*, which strictly avoid causality violations; and *optimistic*, which allow violations and recover from them. In the past three decades, numerous approaches have been proposed by different researchers in this field. A number of surveys can be found in the literature which summarize both conservative and optimistic techniques [1–5]. This paper reviews the classical and recent efforts in the field of parallel and distributed synchronization mechanisms. The paper is organized as follows. Section 2 introduces the conservative synchronization method by classifying different conservative techniques into four categories: deadlock avoidance, deadlock detection and recovery, synchronous operation, and conservative time windows. It also surveys a variety of conservative protocols, comparing them with one another, and presenting various extensions to some of the researches in the area. Section 3 covers the optimistic synchronization method by introducing the key concepts and techniques proposed in the past three decades, with an emphasis on essential topics such as memory management, cascaded rollback, event management, and dynamic process migration. Section 4 discusses recent efforts toward PDES on emerging platforms such as Web services, clouds, and heterogeneous multicore processors. Section 5 presents some of the popular PDES environments followed by a summary of hybrid synchronization techniques. Section 6 concludes the paper with a discussion on future research directions.

2. Conservative synchronization algorithms

As discussed in the previous section, conservative synchronization algorithms strictly avoid any occurrence of causality errors. To do so, the LP is blocked from further processing of events until it can make sure that the next event in its local Future Event List has timestamp smaller than the arrival time of any event that might be arriving at the LP in future. The main issue of any conservative parallel simulator is determining if it is safe for a processor to execute the next event. To deal with this issue, several techniques have been proposed which are further classified into four categories: methods with deadlock avoidance, deadlock detection and recovery, synchronous operation, and conservative time windows.

2.1. Synchronous operation

The first techniques developed for solving these problems proposed different centralized and decentralized mechanisms for implementing global clocks, and they used synchronous operations for the parallel discrete-event simulations. In [6] the authors proposed a centralized mechanism with one dedicated processor controlling a global clock (which represents the global virtual time of the simulation). Under that scheme, all the LPs' local clocks are kept at the same value at every point in real time, and the simulation proceeds according to this global clock, which is advanced based on the minimum timestamp of all possible next events. Peacock et al. [7] introduced the distributed implementation of such a global clock, which was used by [8] on a hierarchical LP structure to determine the minimum next event time. The *min-reduction operation* [9] used a hierarchical LP organization. In this method, the minimum timestamp is moved to the root of a process tree, and it is then propagated down the tree. The *Distributed Snapshot Algorithm* [10,11] proposed a method to avoid the bottleneck of a centralized global clock coordinator by enabling the processes to record their own states and the states of the communication channels. In this way, the set of process and channel states recorded conform a global system state.

Three efficient algorithms for global snapshots in large distributed systems are presented in [12,13]. The proposed algorithms (a grid-based, a tree-based, and a centralized) overcome the issue of scalability of other existing global snapshot algorithms. Experiments showed that the proposed mechanisms significantly reduce the message and space complexity of a global snapshot.

In general, synchronous protocols decompose the simulation into two phases: (1) processing safe events, (2) performing global computations to determine such events. Unlike the detection and recovery methods that will be discussed in the following sections, synchronous mechanisms are deadlock-free. However, they continuously suspend and restart the simulation. In contrast, a major disadvantage of detection and recovery method is that during the period leading up to a deadlock, the execution may be largely sequential, leading to limited speedup.

2.2. Deadlock avoidance

The first asynchronous parallel simulation protocol was a conservative technique developed independently by Chandy and Misra [14], and Bryant [15]. In the *CMB Chandy–Misra–Bryant* (CMB) algorithm, LPs are assumed to be connected statically via directional links. LPs communicate through timestamped messages, also called *event messages*, which are transmitted from one LP to another, in non-decreasing timestamp order. This guarantees that the timestamp of the last message received on an incoming link is a lower bound of any future event messages that will be received later. At each LP, there is a queue associated with each incoming link that is used to store incoming messages in FIFO order. Each link has its own clock which is equal to the timestamp of the first message in the queue (if there is one), or the timestamp of the last received message (if the queue is empty). The LP repeatedly selects the queue with the smallest clock and, if the queue is not empty, processes the first message available. If the queue is otherwise empty, the LP blocks until a message arrives at the queue, which updates its clock value of the incoming link. Afterwards, the LP selects a new queue with the smallest clock and the procedure is repeated.

Since an LP may block on an empty link, deadlocks may occur in the case of a waiting cycle. The CMB mechanism avoids deadlocks by introducing the notion of *null messages*, which are for synchronization purposes only and do not represent real activities in the model. A null message is a promise about the earliest message that will arrive in the future. When an LP receives such a null message, it advances the clock value associated with the link, and, if possible, it progresses by processing events that are waiting in other queues. If processing is not possible, it propagates the time carried by the null message and other time advancements to its successors by sending out more null messages through its outgoing links. The essential part of this mechanism is determining the null message timestamp. This *lookahead* value defines the degree to which LPs can look ahead and predict future events.

Since each incoming link defines the lower bound for the next unprocessed event, a good measure for the lookahead value can be the minimum among all incoming links' clocks plus the LP service time. In fact, the lookahead represents a lower bound on the timestamp of the next outgoing message. Every time the LP finishes processing an event, it sends a null message on each of its outgoing links to report this bound. When an LP receives a null message, it calculates a new bound based on the information it receives and passes it to its neighbors, and so on. The lookahead can also be determined by the programmer statically. It has been shown that the larger the lookahead, the better is the performance of the algorithm [5,16].

The conventional null-message approach to resolving the deadlock problem in conservative simulation may lead to a livelock if lookahead cannot be guaranteed. Livelock can occur if the cycle lookahead sums to zero. A number of solutions have been proposed to resolve deadlock and livelock problems. For instance, a priority assignment method was proposed in [18] resolving the deadlock problem. With such a scheme, a higher priority is given to the execution of departure event whenever a departure time is equal to a null message indicator. Using this priority scheme in the event scheduling, a process will proceed with the simulation instead of keep sending null messages to other processes.

In order to increase the set of safe events, a global reduction computation can be used to derive a *Lower Bound on the Timestamp* (LBTS) among the events that can be received by a LP in the future (i.e., the minimum timestamp of the next future event in the entire simulation system). With such information, each LP can safely process any pending events with a timestamp smaller than the LBTS value [4,53]. One of the challenges of time synchronization protocols is the *transient message* problem [2]. A transient message is a message that has been sent but not yet received by the intended recipient. Fig. 2 illustrates the transient message problem.

Transient messages are a problem if asynchronous message sends are allowed; that is, the sender is permitted to continue execution after sending a message to another processor without waiting for an acknowledgment from the recipient of the message. The LBTS protocol deals with transient messages by defining the LBTS to be the minimum timestamp on any

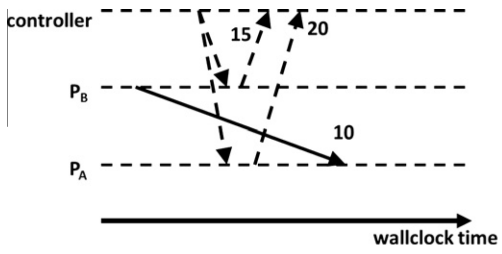

Fig. 2. Transient message problem. P_A needs to take into account the transient message (with timestamp 10) coming from P_B, which is received after a later message (with timestamp 15) had arrived from P_B.

outgoing message that can possibly be received in the future. Another solution to transient message issue is using message counters, where each processor maintains two local counters indicating (1) the number of messages it has sent, and (2) the number of messages it has received. A system is said to be transient message-free if (1) all of the processors have reached a point where no more new messages are produced, and (2) the sum of all of the send counters across all of the processors is equal to the sum of the receive counters across all of the processors.

2.2.1. Variations of CMB

The CMB algorithm can produce many null messages, degrading the performance of the simulation. Since its original implementation, numerous approaches have been proposed aiming at reducing the number of null messages. Here, some of the variations on the CMB approach that deal with this issue are presented.

The SRADS (Shared Resource Algorithm for Distributed Simulation) protocol by Reynolds [17] was the first *on-demand* null message-based method designed primarily for deterministic models where potential message arrival times are predictable. In SRADS, when a receiving link with the smallest timestamp runs out of messages to process, which indicates a process is about to block, a *request next message* is sent only to the link at the sending side of the link. This, however, introduces performance limitations since the time taken to receive a null message by request is doubled due to request transmission and send transmission. Another on-demand null message technique is the *demand-driven* null message protocol [19], which avoids the aggressive distribution of null messages by enforcing LPs to send null messages only when they are asked to. All synchronization messages are of fixed size and independent of the number of processors.

Misra [20] and Peacock et al. [21] also revisited the CMB mechanism by imposing the idea of sending null messages on demand rather than after each event. Nicol and Reynolds [22] used a variation of this approach for distributed simulations with shared resources. Su and Seitz [23] investigated a family of variants of the basic CMB algorithm to speedup gate-level simulations on an Intel iPSC computer. They focused on reducing the volume of null messages by deferring sending outputs and packing the information into fewer messages.

Other approaches to null message generation, including *generation after a time-out* and *generation using stimulus nulls* were introduced in [24]. The purpose of the null message after a time-out algorithm is to reduce the system overhead of processing null messages by reducing the actual number of null messages transmitted between LPs. The null messages are transmitted only after a specified amount of real clock time, the *time-out* value. It was shown that when the time-out value increases, fewer null messages are generated, thus reducing overhead. In contrast to the null message with time-out algorithm, the stimulus null variation added, rather than eliminated, null messages. Stimulus null messages are generated and transmitted after the execution of a given number of internal events, specified as a ratio between the events and the stimulus nulls. These nulls are in addition to any nulls normally generated, and they give the receiving LPs an earlier indication of time progression (when compared to the original CMB null message algorithm). Consequently, there is a greater potential to execute the simulation faster.

Although demand-driven protocols reduce the amount of null message distribution, in return, the delay associated with receiving null messages increases because two messages are required. The *carrier-null message* algorithm introduced by Cai and Turner [25] reduces the number of CMB null messages and it increases the lookahead ability by exploring the simulation network topology. A carrier-null message includes extra information, in particular, the message route. This carrier information is a record of all LPs visited by the carrier-null message since its creation. This information allows individual LPs to advance their simulation clocks while keeping the null message traffic low. The carrier-null message scheme only supports simulations with certain communication graphs such as those with nested cycles. Wood and Turner [26] extended the carrier-null message approach by proposing a generalized carrier-null method to support arbitrary graphs. In [27] the *null message cancellation* protocol was investigated, and the impact of the cancellation of spare null messages was examined. Under this protocol, a null message is discarded before being receipt if it is overrun by a message with a larger timestamp. The empirical results showed how the impact of null message cancellation is affected by the lookahead of the LP. Porras et al. [28] improved the CMB algorithm by using null message cancellation, simulation loop optimization, and multicasting techniques. Their algorithm, named *Simloop* reduces the number of null messages and improves the execution of messages by allowing simulation of multiple messages instead of a single message.

Naroska and Schwiegelshohn's *Critical Process First* (CPF) algorithm [29] is also a non-blocking CMB-based technique that is especially well-suited for simulating complex VLSI designs with many logical processes. The algorithm avoids deadlock by repeatedly sending lookahead information about critical processes to other nodes. A process is called critical if it directly affects one or more processes residing on another node. In order to reduce communication overhead, the CPF technique assigns execution priority to the events which may influence critical processes. Higher performance was reported with CPF algorithm compared to deadlock avoidance approaches without priority handling.

The CMB-SMP (Shared memory Multiprocessors) protocol [29], is another extension to CMB asynchronous simulation protocol. It is an efficient protocol that is customized for shared memory multiprocessor systems. To improve performance, simulation objects are partitioned to form LPs with the objective of achieving load balancing and maximum lookahead (between LPs). These LPs are then mapped to the processor and stay at the same processor throughout the simulation. However, this particular approach does not guarantee good performance, especially when the model exhibits dynamic behavior. The dynamic nature of the model makes load estimation difficult and in turn restricts the performance of the parallel simulation.

The *Critical Channel Traversing* (CCT) algorithm [30] extended the CMB algorithm with the addition of rules that determine when to schedule an LP for event execution. CCT attempts to schedule the LPs with the largest number of events that are ready to execute. This is accomplished through identifying critical channels. The CCT algorithm was implemented along with a simulation kernel called *TasKit*, designed for high performance simulation on small to medium sized shared memory multiprocessors. The algorithm provides multi-level scheduling by allowing scheduling large grains of computation even in very low granularity models. In [31], two new versions of the CCT algorithm were presented. The first one, called *simple sender side CCT*, differs from the original in the elimination of busy waiting. Consequently, it avoids the performance problems that can be caused by busy waiting. The second algorithm, called *receive side CCT*, uses a different strategy for updating channel clocks and scheduling objects connected to critical channels. Receive side CCT reported better scaling with respect to the connectivity of the model, but at the cost of additional overhead for low connectivity models.

Boukerche and Das [32] proposed a null message algorithm that reduced the overhead of null messages using load balancing. The synchronization protocol is a variation of CMB null messages combined with a load-balancing algorithm that assumes no compile time knowledge about the workload parameters. The algorithm is based on a process migration mechanism, and the notion of the *CPU-queue length*, which indicates the workload at each processor. In addition, they presented two variations of the algorithm: a centralized, and a multi-level hierarchical method.

Other null message reduction algorithms that have been proposed use a generic mathematical model to approximate the optimal values of the parameters that are directly involved in the performance of a time management algorithm [33]. Thomas et al. [34] proposed another null message reduction algorithm based on grouping and status retrieval by determining an optimum value of the lookahead.

There have been varied efforts trying to improve the lookahead computation. For example, in [35] the authors presented a method where the compiler automatically extracted information about the lookahead present in the application. The *lock-free* algorithm [36] is another conservative scheduling technique implemented for shared-memory multiprocessor machines, which uses *fetch&add* operations to avoid the inefficiencies associated with using locks. The authors show that compared with lock-based scheduling algorithms, the lock-free algorithm exhibits better performance when the number of logical processes assigned to each processor is small or when the workload becomes significant. However, due to the overhead spent for extra bookkeeping, only modest performance gain is achieved for a large number of logical processes. Solcany and Safarik [37] presented a user-transparent conservative parallel simulator that allows users to build simulation models with lookahead transparently. To do so, they analyze the conditions for cumulating the lookahead of entities inside the same LP, and using this information they derived a mechanism to calculate such cumulated lookahead based on the Dijkstra's shortest path first algorithm. Chung and Kyung [38] proposed a scheme for the prediction of the software execution path in order to extend the lookahead computation for parallel multiprocessor simulation. They used templates for predicting the program execution path, which are generated by software analysis. Then, a processor model obtains the lookahead by evaluating the templates at simulation time. The proposed method aggressively extends the lookahead of null messages by executing the path prediction of the software application dynamically.

Other studies have devised and compared variants to the CMB algorithm by evaluating the performance of the algorithms for inefficiencies and overhead. Curry et al. [51] studied the performance of systems using a sequential Centralized Event List (CEL) and compared the results with those of CMB. In their experiments, the performance of a CMB-based system was compared with three CEL implementations namely the heap, splay tree, and calendar queue for a particular workload model. The results showed that both the number of instructions executed and the cache behavior have significant impact on the performance, and the superior cache performance was able to make up for a larger number of instructions executed. The performance of a CEL-based parallel discrete-event simulator was studied using a benchmark called DEVStone [187] that generates a suite of models with varied structure and behavior automatically. Park et al. [39] compared the performance and scalability of a *lazy null message* algorithm with global reduction approaches. They suggested that, for scenarios simulating scaled network models with constant number of input and output channels per LP, the lazy null message algorithm offers better scalability than efficient global reduction based synchronous protocols. Bagrodia and Takai [40] studied the performance of three diverse conservative algorithms implemented in Maisie: a synchronous algorithm (conditional event), an asynchronous algorithm (with null messages), and a hybrid algorithm (ANM – *Accelerated Null Message*) that combines features from the preceding algorithms. Maisie models were developed for standard queuing network benchmarks, and various configurations of the model such as model connectivity, computation granularity, load balance, and lookahead were executed using the three different algorithms. Song et al. [41] discussed an empirical study of conservative scheduling by examining several heuristics that help in identifying critical events. They presented a performance study comparing several scheduling algorithms based on an LP's next event timestamp, safe time, or local simulation clock. In [43], a performance evaluation of a CMB protocol was investigated. They analyzed the performance and behavior of each logical process, and showed that, in the same simulation, different LPs can show different performance. The analyses were performed by adding software monitors to the simulation code. The monitors computed some metrics whose values were used to estimate the performance of each logical process in execution time. Jafer and Wainer [42] proposed a variation of the CMB algorithm, called, Global Lookahead Management (GLM) protocol for calculating a global lookahead. The GLM protocol which borrows the idea of safe processing intervals from the conservative time window algorithm and maintains global synchronization in a fashion similar to the distributed snapshot technique. Under the GLM scheme, a central lookahead manager exists on LP0 which is in charge of receiving every LP's lookahead, identifying the global minimum lookahead of the system, and broadcasting it via null messages to all LPs. The simulation is divided into cycles of two phases: Parallel phase and Broadcast phase. This implies that the LPs are no longer required to send their lookahead information directly to each other; rather, they send their lookahead via null messages to the LM. Liu and Li [180] examine a parallel processing method for simulations of large-scale networks using a hybrid model. Their method benefits from the observation that the time it takes to propagate fluid characteristics along the path taken by the traffic flows has a lower bound equal to the minimum link delay computed using the ordinary differential equations (ODEs). Thus, better lookahead can be achieved compared to that of the corresponding discrete-event packet-oriented model.

2.3. Deadlock detection and recovery

Another approach for conservative synchronization is to allow deadlocks to occur, and to provide a mechanism to detect and recover from them. This approach eliminates the use of null messages and the overhead associated with their communication traffic. Deadlock is broken by allowing to processing the event with the smallest timestamp. Chandy and Misra [44] proposed an asynchronous distributed simulation approach via a sequence of parallel computations. Their approach did not use a global clock nor did they use a single process to drive the simulation. Rather, to avoid bottlenecks, they use a special process, called the *controller*, which synchronizes the LPs when the simulation deadlocks. Under that scheme, the simulation is divided into a sequence of computations: the *parallel phase* and the *phase interface*. The controller is only responsible for detecting the termination of one phase and initiating the next one.

One approach to determine safe events is to perform a set of distributed computations across all the LPs. The *Critical Path Analysis* algorithm (CPA) [44,45] generates an acyclic event-dependency graph by tracing the events in the simulation. The critical path is calculated as the longest event path in the event graph, and its related time is considered as the lower bound on the execution time of the simulation. Srinivasan and Reynolds [46] mention that the conservatism of the CPA is relaxed in the sense that it only considers the dependency among events, requiring each LP to know exactly what the next event is, and when does it arrive. This is not ideal since events in a parallel simulation are unpredictable at runtime. The *State Causality Analysis* algorithm (SCA) [46,47] overcomes the limitation of CPA by focusing on the dependency of the logical process states, rather than on unpredictable events. This technique takes into consideration the effect of many algorithm independent factors, such as lookahead, I/O overhead, physical transfer delay, processor speed, and event distribution.

Groselj and Tropper [48] proposed the *time-of-next-event* (TNE) algorithm for situations where multiple LPs reside on a single processor. TNE relies upon a shortest-path algorithm and increases parallelism by computing the largest lower bound of all LPs independently on every processor. The advantage of this approach is that it does not rely on message passing to distribute the lookahead information; rather, the algorithm is executed on each LP independently. A deadlock recovery algorithm is used to resolve inter-process deadlocks. Boukerche and Tropper [49] presented an extension to TNE, namely SGTNE (Semi Global TNE), whose goal was to exploit lookahead information from both the local and the neighbor LPs (unlike TNE where only LPs within a process are used to unblock an LP). Consequently, SGTNE outperforms TNE, as it allows a higher degree of parallelism as well as avoiding inter-process deadlocks.

2.4. Conservative time windows

Lubachevsky [52] was the first to introduce the idea of a *moving time window* to determine the set of safe events that can be executed in parallel. Using this approach, a lower edge is defined for the window, based on the minimum timestamp of all the unprocessed events, and a window size. Any event whose timestamp is within the window size is eligible for processing. Although this mechanism eliminates the overhead associated to the search for safe events, an important success factor is the window size. A small window size would decrease parallelism while a large window size would result as if there was no time window at all. An appropriate window size can be obtained either from the programmer, the compiler, or at runtime by monitoring the simulation [1]. The *Moving Time Windows* (MTW) protocol [53] is a relaxed version of Lubachevsky's approach where global windows are adjusted dynamically and the events within a window are assumed to be parallel. When an event with a timestamp earlier than the LP's clock is received, an anomaly occurs. Ayani and Rajaei [54] show that better parallelism can be achieved using the *Conservative Time Window* (CTW), where the global ceiling of the window is eliminated, and allowing different windows to have different sizes.

Lemeire and Dirkx [55] proposed a hybrid synchronization technique that combined the asynchronous CMB algorithm with CTW to maximize the lookahead capabilities of a model by using lookahead accumulation. The algorithm exploits maximum performance by accumulating lookahead information, and computing it using the global lookahead of the multiprocess (LPs residing on a processor). Lin et al. [56] presented a method called *micro-synchronization* to exploit the parallelism inside each LP. Unlike the methods of lookahead accumulation [55] and local time warp [57], this technique keeps the traditional use of lookahead among LPs unchanged, while imposing a relaxed sequential event scheduling inside each LP, which can statistically increase the lookahead.

2.5. Other conservative protocols

Numerous conservative protocols have been proposed, and some of them are presented here. Several of these protocols were defined as a combination of synchronous approaches with event-driven clock progression. The idea is to divide the computation into cycles, in which one first determines the *safe events*, and then processes all those events. Ayani [58] used the concept of distance between LPs to determine the safe events. Under this scheme, the distance gives the minimum time it takes for an event in one LP to directly or indirectly affect another LP (similar to a shortest-path algorithm). This draws a bound on when should an LP expect an event from its neighbors. Prior to Ayani's work, Lubachevsky proposed the *bounded-lag* algorithm [52] which also took advantage of the propagation delay between LPs to exploit lookahead [59]. The algorithm used a time interval (called the time lag) in order to compute a set of LPs that can affect a given LP within the lag interval.

The *conditional event* protocol [60] categorized events into two types: *definite*, and *conditional*. Definite events are scheduled locally, while conditional events require communication among all LPs to determine the earliest conditional event globally (which is then converted into a definite event). Nicol [61] used a similar idea based on *synchronization barriers* and introducing time windows with the restriction that all events within a window are safe to process. Similar to the conditional event approach, global computations are conducted to determine the time of next synchronization point.

Nicol and Liu [62] proposed a composition strategy by combining the synchronous barrier synchronization (global) with channel scanning (local) synchronization protocols to allow tailoring the synchronization mechanism to the model being simulated. Their attempt to combining synchronous and asynchronous approaches allows using one method in part of the model where the other method is weak. Using this approach, the effect of high connectivity is limited by making most of a node's channels synchronous. On the other hand, by making channels with low lookahead asynchronous, the effect of unusually low lookahead is limited.

The success of all the conservative synchronization algorithms presented in this section largely depends upon the ability to predict the future, in terms of the lookahead or LBTS [1]. In order to achieve acceptable performance, this, in turn, requires an effective use of *application-specific* information such as the topological structure of the network of LPs, the characteristics of the communication network, and the underlying model behavior. A side effect of this requirement is that a seemingly minor changes to the model could affect the simulation performance dramatically, hindering the robustness of the application [2]. Perhaps the most prominent drawback of conservative approaches is that they often cannot fully exploit the potential parallelism available in a simulation, especially when the estimated lookahead or LBTS values are overly pessimistic and when global synchronizations are performed too frequently in the synchronous execution mode. The optimistic synchronization algorithms introduced in the following sections do not have any of these problems. Nonetheless, when the application characteristics are favorable, conservative approaches can reduce the execution time significantly with moderate memory consumption (see, e.g., [63,64,38]).

3. Optimistic synchronization algorithms

Jefferson's Time Warp (TW) mechanism [65] was the first and remains the best known optimistic synchronization protocol. A TW simulation uses virtual time to model the passage of time in a simulation, and it is driven by a set of Time Warp Logical Processes (TWLPs), each of which has its own Local Virtual Time (LVT) and processes events autonomously without explicit synchronization. TWLPs differ from ordinary LPs, such as those used in sequential and conservative simulations, in the way in which the states and events are managed. Specifically, an ordinary LP maintains only one copy of its state (i.e., its current state), which is updated repeatedly during the event execution. Furthermore, an ordinary LP does not need to keep a record of past input and output events, allowing the events to be reclaimed immediately after execution. In contrast, each TWLP needs to manage a history of its past events (both input and output) and states. This includes three data structures: an input queue that contains input events that are recently arrived from the other TWLPs (sorted in receive timestamp order), an output queue that holds *anti-messages* that are negative copies of the recently sent output events (sorted in send timestamp order), and a state queue that stores the recent states of the TWLP. As will be discussed shortly, the historical data saved in these queues cannot be discarded until it is guaranteed that no event with a smaller timestamp can ever be received by any TWLP in the system. Hence, in a TW simulation, the events and states are considered as persistent in the sense that they continue to exist in the queues for a while after having been processed or updated by the TWLPs.

A *causality error* in these algorithms is manifested by the arrival of an event with a timestamp (i.e., receive time) that is less than the LVT of the receiving TWLP. Such an event is called a *straggler* event. TWLP recovers from the causality error by undoing the effects caused by those events processed speculatively during the previous computation. This recovery operation is known as *rollback*, during which the state of the TWLP is restored to the one that was saved just prior to the virtual time as indicated by the straggler's timestamp.

Since false messages (i.e., those generated during the speculative event processing) may have spread to other TWLPs, they must be cancelled as well. Cancellation of false messages is achieved by sending the anti-messages previously stored in the output queues. When an anti-message encounters its positive counterpart in a TWLP's input queue, they *annihilate* each other immediately, thus cancelling the positive one. If the false message has already been processed before the arrival of the anti-message, the destination TWLP is also rolled back, leading to further propagation of anti-messages in the system. The rollbacks caused by straggler events are referred to as *primary* rollbacks, while those triggered by anti-messages are known as *secondary* rollbacks. The timestamp of the straggler or anti-message is commonly called rollback time. The rollback-handling algorithms constitute the local control mechanism of the TW protocol.

The TW protocol also includes a global control mechanism that requires a distributed computation involving all of the TWLPs in the simulation system to handle such global issues as memory management, I/O operations, and termination detection. A key concept behind the global control mechanism is the Global Virtual Time (GVT), which is defined as follows [1].

Definition: Global Virtual Time at wall clock time T (GVT_T) during the execution of a TW simulation is defined as the minimum timestamp among all unprocessed and partially processed messages and anti-messages in the system at wall clock time T.

It has been shown that GVT never decreases, even though the LVTs can be reset frequently during rollbacks [65]. That is, any TWLP will never receive an event with a smaller timestamp than the current GVT. Moreover, the GVT deals with transient messages in the same fashion as in the conservative LBTS protocol (see Section 2.5). Therefore, all but the last events and states saved before the GVT can be discarded safely through a procedure known as *fossil collection* to free up the memory occupied by these historical data. In addition, I/O operations scheduled before the GVT can also be committed irrevocably. Note that the global control mechanism must estimate GVT and perform fossil collection every so often to reduce the possibility of memory stalls, where the simulation cannot complete because of memory exhaustion. Since the GVT estimation incurs a significant overhead in terms of processor time and network bandwidth, a trade-off between TW execution efficiency and memory space usage needs to be sought in choosing the frequency of GVT computation [65].

Fujimoto summarized a few advantages of optimistic synchronization algorithms over conservative techniques [50]. First, optimistic algorithms can generally exploit higher degrees of parallelism, while conservative techniques tend to force a sequential execution when it is not necessary, especially for models that exhibit relatively small lookahead values. Secondly, unlike conservative techniques, optimistic algorithms are less reliant on application-specific information for correct execution, even though execution efficiency can be improved if such information is available. Thirdly, since events are processed in a non-blocking manner, optimistic algorithms do not suffer from the deadlock problem as asynchronous conservative algorithms do. On the other hand, the persistent storage of historical data and the possibility of cascaded rollbacks in optimistic simulations could result in performance degradation to a certain extent. In a recent study, Carothers and Perumalla compared the relative performance of conservative and optimistic PDES on a Blue Gene/L supercomputer using up to 16,384 processor cores [66]. Through quantitative evaluation of different factors (such as lookahead, event timestamp distribution, and processor core counts), this study demonstrated that conservative execution can attain better performance for models with large lookahead on up to 8192 cores, whereas optimistic execution prevails in cases with lower lookahead and on larger number of cores, indicating better resilience to model lookahead values and scalability with increasing number of processors.

The TW protocol has been employed in many real-world applications, achieving significant speedups in simulations of communication networks [67], battlefield scenarios [68], biological phenomena [69], and computer systems [70].

In order to reduce operational overhead and improve the performance of TW-based optimistic simulations, a wide variety of techniques and optimization strategies have been proposed. While these results are very encouraging, several challenging issues remain to be resolved to meet the ever-increasing demands and performance requirements in large-scale TW simulation of complex systems. This section evaluates some of the most relevant previous contributions made towards this goal, with an emphasis on memory management, cascaded rollback, event management, and dynamic process migration in TW simulations.

3.1. Memory management

It is well known that TW-based optimistic parallel simulation requires more memory space to execute efficiently than an equivalent sequential simulation [71]. This additional memory space requirement stems mainly from the need for saving historical input/output events and states in the persistent queues during a simulation. Different memory-conserving techniques have been proposed to reduce memory consumption in TW simulations. One challenge, though, is to maintain high-performance TW execution without undue overhead even when the system memory is tight.

3.1.1. Fossil collection algorithms

Fossil collection is one way to reduce the possibility of memory stalls in TW simulations. To achieve efficient fossil collection, different approaches have been attempted. For instance, Young et al. proposed an optimistic fossil collection technique that allows each TWLP to make its own fossil collection decisions based on locally predicted information without estimating the GVT globally [74]. Introducing optimism to fossil collection comes with a risk in that a TWLP may later be rolled back to a previous state that has already been reclaimed in the speculative fossil collection. To make this technique feasible, a recovery mechanism is used to reconstruct the erroneously reclaimed historical data. The recovery mechanism employs a distributed algorithm to create consistent checkpoints during a simulation, saving the states of all processes and inter-process communication channels. While this technique can reduce the cost of GVT computation, it entails extra overhead for risk prediction and fossil recovery. Young et al. demonstrated that the optimistic fossil collection could achieve a comparable performance with the GVT-based algorithms [75].

A fossil identification mechanism, proposed by Chetlur and Wilsey [76], also attempts to reclaim fossil data without the need for GVT estimation. This mechanism uses an extended timestamp structure, known as plausible total clock, to record the causal relation between events. Based on this causal information, a TWLP may be able to reclaim certain historical data beyond the current GVT value, thus releasing more memory. However, this approach incurs an additional communication overhead for transferring extra causal information associated with events. Moreover, it assumes a static TWLP interconnection topology with certain characteristics.

Vee and Hsu [77] proposed an enhanced fossil collector, referred to as PAL, which can reduce the cost of fossil collection by prioritizing the TWLPs based on the amount of fossil data they have (thus allowing for more efficient retrieval of fossil data from the TWLPs). The algorithm also concentrates all committed events in a shared data structure so that the memory buffers can be reused later in event-saving operations (thus reducing memory allocation and deallocation overhead).

Memory management using fossil collection alone has two main drawbacks. First, fossil collection still constitutes a significant overhead in large-scale simulations because the operation needs to handle a large number of TWLPs and a great amount of fossil data that are usually scattered across the entire simulation system. Secondly, even frequent fossil collection cannot guarantee the absence of memory stalls if the GVT does not advance sufficiently fast, especially when the simulation needs to execute a large number of simultaneous events at each virtual time.

3.1.2. Memory stall recovery algorithms

Different approaches aimed to recover from memory stalls in TW simulations have been explored, resulting in the development of techniques such as cancelback [71], artificial rollback [72], and pruneback [73]. Both cancelback and artificial rollback require a global pool of memory to be shared by all of the TWLPs in the system. All memory requests are granted from the pool, and the memory released is returned to the pool. They differ in how to deal with situations when the pool runs out of memory. Under the cancelback mechanism, some future pending events are returned to their original senders, forcing the sender TWLPs to roll back and release memory. With artificial rollback, the TWLPs with the greatest LVTs (i.e., the most aggressive ones) are rolled back artificially, releasing memory in the process. The need for a common memory pool, nonetheless, makes these two approaches best suited for shared-memory architectures. On the other hand, the pruneback mechanism does allow for recovery from memory stalls on distributed-memory multiprocessors. Instead of using rollback as the means to memory reclamation, the pruneback mechanism releases the memory occupied by past states in the state queues, producing an effect that is similar to infrequent state-saving strategies. This technique thus incurs a cost similar to infrequent state saving as well. That is, a TWLP may have to roll back further in the past than is necessary, and resume forward event processing from there (an action called *coast forward* in the PDES literature). In addition, the pruneback mechanism targets only past states, while the memory used by past input/output events remains unaffected. All these techniques attempt to recover from memory stalls at the expense of a time penalty, which can be very high in certain circumstances.

3.1.3. Checkpointing algorithms

Instead of trying to recover from memory stalls, an alternative approach is to save fewer historical data in the first place. To this end, different checkpointing algorithms have been proposed to reduce state-saving overhead. Some of them, known as *infrequent state-saving* or *periodic state-saving* techniques, focus on reducing the number of states saved in a simulation (see, e.g., [78]) Others, known as *incremental state-saving* techniques, try to reduce the amount of data that need to be saved in each state (see, e.g., [79]). There are also techniques that multiplex different state-saving mechanisms to improve performance (see, e.g., [80]). While these techniques can reduce the state-saving overhead, they increase the computational cost in one form or another. For instance, infrequent state saving requires an extra coast-forward operation with a higher rollback overhead, while incremental state saving needs to keep track of the changes made to individual state variables with an increased event-processing overhead. Moreover, incremental state saving can improve simulation performance only when a small portion of the state data is subject to modification during each event execution, making it mainly suitable for simulations such as digital logic circuits [79].

3.1.4. Other memory conserving techniques

Other techniques have also been investigated to reduce memory consumption. A relatively new technique for conserving memory in optimistic parallel simulation is called *reverse computation* [81], which allows the LPs to restore their states by computing the inverse operations for each event being rolled back. It has been shown that reverse computation can achieve a significant performance improvement in such simulations as queuing network models [81], personal communication service networks [82], and physical systems [83]. This technique, however, usually requires annotation and manipulation of the source code at the individual statement level, making it difficult to modify model logic. Although this issue can be alleviated by using advanced code transformation and compilation tools to generate reverse code automatically, certain destructive operations (which result in the loss of data) might not be perfectly reversible (e.g., certain bit-wise and floating-point operations) [84], thus limiting the applicability of the technique. Furthermore, reverse computation potentially increases the computational cost during rollbacks, especially when a large number of events need to be processed reversely at the LPs. In a case study, Perumalla and Seal [183] analyze rollback in a massively parallel optimistic execution of epidemic outbreak by combining reverse computation with a small amount of incremental state saving, achieving a speedup of over 5500 and other runtime performance metrics.

The *event reconstruction* technique proposed by Li and Tropper [85] intends to reduce the overhead associated with event saving. Based on the observation that the size of events can be very large in some cases, they suggested a method for reconstruction of both input and output events by comparing the differences between adjacent states saved in the state queues. Significant performance gain has been reported for VLSI simulations, yet it is recognized that this method works well only in a certain class of simulations with fine event granularity and small state size.

In [86], the authors proposed a user-controlled state-saving mechanism that allows for efficient and flexible checkpointing at runtime. With this mechanism, a TWLP can make its own state-saving decisions on an event-by-event basis using application-level knowledge. A specific type of user-controlled state-saving, referred to as the *Message Type-based State-Saving* (MTSS) strategy, was implemented in the PCD++ simulator, so that a TWLP saves its state only for a certain type of events in DEVS (Discrete Event System Specification) based simulations. This approach reduces memory consumption and state-saving overhead, while avoiding the need for coast-forward operations during rollbacks. Unlike other infrequent state-saving techniques, no performance penalty is incurred as the result of saving fewer states. The MTSS strategy was further optimized in the Lightweight Time Warp protocol [87], which takes advantage of the specific computational properties of DEVS simulation to improve performance [88,177].

3.2. Cascaded rollback

Rollback propagation can have a significant impact on the performance of optimistic PDES systems. As mentioned earlier, two types of rollbacks may occur in a TW simulation, namely primary rollbacks and secondary rollbacks. An optimistic synchronization protocol is said to be *aggressive* if primary rollbacks are allowed, while the protocol is considered to allow *risk* if secondary rollbacks are possible. The TW protocol is aggressive and allows risk. In general, both the rollback width (i.e., how many TWLPs are involved in a rollback propagation) and rollback depth (i.e., how many events are unprocessed by a TWLP during a rollback) cannot be bounded easily. Without care, this can lead to uncontrolled rollback behavior commonly known as *domino effect*, which jeopardizes the stability and scalability of the entire system. A major cause of the domino effect lies in the need for cancellation of many false messages during secondary rollbacks, which give rise to cascaded rollback where a large number of TWLPs are involved in the propagation [1].

3.2.1. Optimism control mechanisms

One approach to rollback reduction is through optimism control, which tries to regulate overly optimistic execution. The *Moving Time Window* (MTW) algorithm, which puts a bound on the difference in the LVTs of the TWLPs, is an early example of this approach [53]. Many other techniques for optimism control have been developed over the last two decades. Examples include the Breathing Time Warp protocol [89], the Global Progress Window algorithm [90], the Elastic Time algorithm [91], and the Switch Time Warp mechanism [92]. Likewise, varied flow-control and learning based algorithms have been defined (see, e.g., [93,94]).

In addition, different adaptive algorithms have been used to improve simulation performance by adjusting specific control parameters dynamically at runtime to influence the degree of optimism (see, e.g., [95,96]).

The basic idea behind all these optimism-limiting techniques is to improve event *temporal locality* in a TW simulation so that most of the events processed concurrently have relatively close timestamps. In doing so, these techniques sacrifice the degree of parallelism to a certain degree. Moreover, these techniques often rely on knowledge of certain aspects of the global simulation state in order to tune the control parameters, incurring an extra overhead for information collection and analysis during the simulation.

3.2.2. Event cancellation algorithms

Rollback efficiency can also be improved using different cancellation mechanisms. The original TW protocol adopts an aggressive cancellation scheme that sends anti-messages immediately when a TWLP is rolled back. The *lazy cancellation* mechanism, originally proposed by Gafni [97] and subsequently analyzed by Lin and Lazowska [98], is one of the first attempts to reduce the communication overhead of event cancellation. With this mechanism, the sending of anti-messages at a TWLP is suspended until its necessity has been verified when events are reprocessed after a rollback. However, cancellation of false messages may be delayed as a result; and additional computation and memory space is required to realize the lazy cancellation algorithms.

A *throttled lazy cancellation* scheme has been proposed recently to slow down the spread of potentially incorrect computation when events are re-evaluated during lazy cancellation operations [99], but only at the expense of increased communication cost for broadcasting special control messages to block and unblock the TWLPs. This is similar to the mechanism used in the Wolf Calls protocol originally proposed by Madisetti et al. to contain error propagation in TW simulations [100]. Furthermore, broadcasting control messages is not without limitations. For one thing, it may block some TWLPs unnecessarily. For another, the effectiveness of this mechanism depends on the relative speeds at which erroneous computation and control messages may spread.

To further reduce the communication overhead of event cancellation, an optimization strategy called *early cancellation* can be used to cancel false messages in place in the buffer of a programmable network interface controller [101], which demonstrates the potential of using specialized hardware to improve TW performance, but also limits the utility of the strategy in TW simulations on general-purpose computing platforms.

Other studies have shown that cancellation performance can be improved by capturing the causal relationship between events [102]. By exploiting event causal dependency, a *proactive cancellation* mechanism that can be used to prevent cascaded rollbacks was developed [103]. Using a similar strategy, a *batch-based cancellation* algorithm that allows a TWLP to recover from a causality error with at most one rollback was proposed [104]. However, this algorithm introduces extra communication overhead for exchanging causal information and rollback histories between TWLPs, as well as additional computation overhead for reclaiming these data during fossil collection.

3.2.3. LP Aggregation techniques

Different LP aggregation techniques have been investigated to mitigate the overhead of event cancellation. One example is the Local Time Warp protocol proposed by Rajaei et al. [57]. In this protocol, the global simulation space is divided into several sub-regions referred to as *clusters*, each of which contains a set of TWLPs. While the TWLPs within a cluster are executed optimistically (based on TW), the clusters themselves are synchronized in a conservative fashion, thus preventing false messages from propagating beyond cluster boundaries. Consequently, cascaded rollbacks and memory stalls need to be handled only locally. However, the Local Time Warp protocol, like many other hybrid approaches that combine both conservative and optimistic algorithms in a simulation, may suffer from reduced parallelism. Furthermore, it requires careful control to balance the optimistic local simulation of individual clusters with the global virtual time horizon established by the conservative algorithm [105].

The *clustered adaptive-risk* technique, proposed by Soliman and Elmaghraby [106], is another example that aims to control the degree of risk in TW simulations. Similar to the Local Time Warp protocol, the TWLPs are grouped into clusters. Nonetheless, instead of applying a conservative synchronization protocol at the global level, the technique uses an adaptive algorithm to keep the probability of cross-cluster rollback propagation below a user-defined threshold. This is achieved by tuning the intervals between the release times of buffered inter-cluster messages based on observed simulation behavior at runtime (at the cost of additional computation and memory space overheads).

While the above techniques can be used to improve rollback performance, every TWLP in the system is still subject to rollback operations that are triggered either locally or globally, requiring each TWLP to maintain its persistent event and state queues, just like in the original TW mechanism.

To enhance the performance of TW-based digital logic simulations, Avril and Tropper [107] introduced a Clustered Time Warp protocol that uses TW to synchronize clusters of LPs globally, whereas the execution of LPs in each cluster is scheduled sequentially by a cluster environment. The cluster environment uses a time zone table (to detect changes in virtual time), a cluster input queue (to receive events from other clusters), and a cluster output queue (to hold anti-messages that might be sent to other clusters during rollbacks). The rationale behind this approach is that a logic circuit can be partitioned naturally into different functional units, each of which will then be simulated on a distinct processor. Two types of rollback mechanisms are defined in the protocol, referred to as clustered rollback and local rollback. Under the former mechanism, rollbacks are handled at the cluster level. When a straggler or anti-message is received by a cluster, all of the LPs included in the cluster are rolled back together if they have executed an event with a timestamp greater than the rollback time. Although this mechanism can reduce memory consumption (since individual LPs do not need to keep anti-messages in their output queues, and all input events with timestamps greater than the rollback time are discarded during rollbacks), some LPs may be rolled back unnecessarily. Under the latter mechanism, the cluster environment simply forwards the received straggler or anti-messages to the destination LPs so that they can make rollback decisions individually. In this case, the LPs must maintain anti-messages in their output queues; and rollbacks are carried out in the same way as in TW. In the same vein, two checkpointing mechanisms are defined, including a clustered checkpointing mechanism that saves states for the LPs only when remote events are received from other clusters, and a local checkpointing mechanism that saves the state for a LP whenever the simulation time is changed in the time zone table (regardless of whether the time change is caused by a local or remote event). Since both mechanisms use infrequent state saving, coast-forward operations are required during rollbacks with the associated overhead. Using several digital circuit models as benchmarks, different combinations of these rollback and checkpointing mechanisms have been evaluated quantitatively [108]. The experiments showed that, while memory usage can be reduced by up to 40% in some cases, the execution time is comparable or even worse than obtained with the original TW protocol, indicating that a trade-off must be made between execution efficiency and memory conservation.

3.3. Event management

A central issue to be addressed in any discrete-event simulation is event management, which has been studied extensively from different aspects. Here, we highlight previous research related to event set implementation and management of simultaneous events in TW simulations.

3.3.1. Event set implementation

The relative performance of different implementations of event set data structures and algorithms has been a topic of research since the early days of discrete-event simulation. The need for handling potential rollbacks in optimistic parallel simulations makes event management more complex than in a sequential simulation, mainly because past events that have already been processed remain in the event queues. As a result, efficient insertion and retrieval of both historical and future events become necessary for the overall simulation performance [109].

Numerous non-trivial data structures have been investigated in the context of TW-based optimistic simulations. Most of them can be characterized into three broad categories: *list* structures, *tree* structures and *multi-list* structures. Examples of list structures include the Indexed Lists [110] and the SPEEDES Queue [111]. Tree structures are exemplified by Binary Heaps, Skew Heap, and Splay Trees [112], while the Lazy Queue [113], the Ladder Queue [114], and the Calendar Queues [115] are based on multi-list structures.

A primary motivation behind these efforts is to achieve efficient event queue operations as the number of events stored in the event queues increases in large-scale and fine-grained simulations. While these approaches have proven to be quite useful in improving performance, an attractive alternative solution would be to keep the event queues relatively short throughout a simulation.

3.3.2. Simultaneous events

The way in which simultaneous events are managed in discrete-event simulation can have serious implications on simulation correctness, reproducibility, and performance [116]. As noted by Jha and Bagrodia [117], simultaneous events may occur in a discrete-event simulation for three general reasons. First, the physical system may include many independent activities. Consequently, when the system is decomposed into a set of LPs, it is convenient to represent the interactions between different portions of the system as simultaneous events. Secondly, the limited resolution of simulation time can also lead to events with the same timestamp even though these events are not truly parallel in the physical system. Finally, the need for modeling activities with zero delay time (or a delay that is negligible compared to the duration of other activities being modeled) often results in zero-delay LPs that generate output events with the same timestamp as the received input events.

Many studies have been devoted to ordering of simultaneous events (see, e.g., [118]). Two types of tie-breaking mechanisms are commonly used in discrete-event simulation [117]. One of them, referred to as *user-consistent and deterministic*, bundles all of the simultaneous events received by a LP and executes them based on a set of protocol-independent, user-specified rules. The other, referred to as *arbitrary and deterministic*, relies on the simulation protocol to choose a well-defined implicit ordering of the events. Still, some researchers argue that the appropriate way of handling simultaneous events is to take all possible orderings into account when evaluating the simulation results, rather than forcing the users or the protocols to choose an ordering that may not always serve well the intention of the simulation [119]. Various approaches have been taken to implement tie-breaking mechanisms in the context of PDES. Some of them extend the timestamps of event messages to impose a ranking on the simultaneous events for deterministic execution [120], while others employ the concept of aging for the same purpose [118].

Although these techniques provide a well-founded basis for handling simultaneous events in PDES, the performance consequence of processing a large number of simultaneous events at each virtual time in a simulation has not yet attracted enough attention from the research community. This performance issue is especially important in large-scale TW simulations. Without careful design and proper control, the expanded execution of simultaneous events could have a detrimental effect on TW performance in terms of increased overhead for state saving, rollback, fossil collection, and dynamic process migration.

3.4. Dynamic process migration

Dynamic load-balancing algorithms typically rely on the runtime system state information to make decisions regarding the movement of workload from one processor to another during execution. According to Willebeek-Lemair and Reeves [121], dynamic load balancing can be organized as a procedure with four major components, including (1) processor load evaluation; (2) load balancing profitability determination; (3) load migration strategy, and (4) load selection strategy. All of these components have been studied extensively in the PDES literature, leading to the development of a large number of dynamic load-balancing algorithms. An exhaustive review of these load-balancing algorithms is beyond the scope of this article (but see, e.g., [121–125] for related work on this topic).

Instead, the following discussion summarizes some of the efforts that aim to facilitate process migration in TW simulations. While dynamic load balancing is concerned primarily with distributing the workload as evenly as possible among the processors, process migration focuses on the operation of load transfer to achieve a certain load-balancing objective.

The use of an agile process migration mechanism is recognized as crucial to efficient dynamic load balancing in parallel and distributed systems, as it can minimize the communication overhead and the interference with normal system execution [126]. This is even more important in large-scale TW simulations where a potentially unbounded amount of event and state data associated with a TWLP must be transferred between processors. An early work, presented by Reiher and Jefferson [122], employed a phase-based computation model to reduce process migration cost in the Time Warp Operating System. In this model, the life span of a TWLP is divided into multiple phases, each representing a portion of the execution history of the

TWLP during a specific interval of virtual time. These phases can be transferred individually across processors as needed in order to implement a finer-grained load-balancing scheme below the LP granularity. Yet this computation model suffers from increased overhead for message routing and scheduling during both forward execution and rollbacks because each TWLP can consist of many small fragments scattered all over the system.

Different dynamic load balancing and process migration algorithms have been investigated in the SPEEDES simulation framework [124]. These algorithms use a central coordinator to make global load redistribution decisions regularly during a simulation. If load migration is warranted, each pair of chosen nodes at the LP level then handles the actual load selection and transfer operations. With the Breathing Time Warp protocol [89], a parallel simulation is executed in a cyclic fashion. Each simulation cycle starts with a purely optimistic TW execution, but then switches to a risk-free synchronization mode using the Breathing Time Buckets algorithm [127]. After the risk-free execution stage, a new GVT value is computed, followed by the reclamation of historical state and event data. Therefore, the amount of data to be transferred can be minimized if load migration is carried out at the end of a simulation cycle. This data minimization strategy, however, has two main drawbacks. First, the conservative risk-free execution might reduce the degree of achievable parallelism in a simulation. Secondly, the success of this strategy still depends on GVT computation and fossil collection, which, if performed too frequently, could adversely affect simulation performance.

Based on the Clustered Time Warp concepts, Avril and Tropper [128] developed a dynamic load-balancing algorithm that transfers all of the TWLPs included in a cluster as a group. Although this approach makes it easier to implement the load-balancing algorithm since migration decisions are made only for clusters (instead of for individual TWLPs), it reduces the flexibility of the mechanism and shifts some of the responsibilities to the users, who also need to consider load-balancing issues when partitioning the model.

In a more recent study, Li and Tropper argued that the event reconstruction technique, originally intended for memory conservation in Clustered Time Warp, could also be used to facilitate process migration because only the state data associated with a TWLP need to be transferred [85]. Nonetheless, the study did not indicate whether the proposed migration scheme would be realized at the cluster level or at the individual LP level. In addition, this migration scheme can be applied only to models with certain event and state characteristics that motivated the development of the Clustered Time Warp in the first place.

4. PDES on emerging platforms

4.1. Web-based PDES

Distributed simulation technologies employ multiple distributed processors, connected via communication networks, to execute the same simulation run over a geographic area correctly [168]. A focal point of distributed simulation software has been on how to achieve model reuse via interoperation of different simulation components. Indeed, interoperability is the major function of most existing simulation middleware [168]. Web-services technology is highly leveraged, which has proven useful in achieving model and simulation interoperability [169]. Web-services (WS) provide general interoperability standards, enabling deployment of services on a machine and consumed by another via the Web. They fall into two popular classes: SOAP-based WS and RESTful WS [171].

Web services have emerged rapidly and replaced traditional distributed simulation technologies. For example, the new HLA [172] standard is extended with a Web-service interface. In recent years, there have been some studies conducted in the form of surveys of experts from different backgrounds such as the ones described in [173]. Those studies aimed on analyzing a number of issues concerning the current distributed simulation state-of-the-art and research challenges that must be resolved with the purpose of advancing these technologies use particularly outside the defense sector.

Distributed simulation has also been used in the grid environment. For example, DEVS/Grid [121] implements a grid-enabled DEVS simulator following a layered approach. Another grid example is described in vGrid [174], which divides the model into components that can be grouped together to form a virtual computational unit. SensGrid [182] is a simulator of systems that integrates sensors and a grid. In particular, it is an extension of the widely accepted simulation toolkit, GridSim [188], enabling the execution of simulations in a way that users can perform queries on sensor networks.

Other distributed simulation systems made a use of the JXTA standards [174]. JXTA is an open peer-to-peer (P2P) standards developed by Sun Microsystems (now acquired by Oracle). In the P2P systems, simulation messages may go through a number of intermediate machines before reaching their destinations.

SOAP-based Web services (or big Web services) provide a standard means of interoperating between different heterogeneous software applications, residing on a range of different platforms mainly for software reuse and sharing. At present, it is the leading technology for interoperating remote applications (including distributed simulations) across WAN/Internet networks. For example, a new WSDL API has been added to the HLA IEEE 1516-2007 standard, allowing HLA-compliant simulations to be connected via the Internet using SOAP-based Web services.

The Representational State Transfer (REST) Web-services [170] provide interoperability by imitating the Web architectural style and principles. Recently, a RESTful Interoperability Simulation Environment (RISE) [179] was introduced allowing for interoperating heterogeneous simulation models and tools regardless of their underlying technology or algorithms.

Moreover, RESTful Web-services are gaining increased attention with the advent of Web 2.0 and the concept of mashups [175]. A mashup groups various services from different providers and presents them as a bundle in order to provide single integrated service. For example, IBM enterprise mashup solutions [176] aim on integrating Web 2.0 functions as rapid as possible. Nowadays, RESTful Web-services are supported, in conjunction with SOAP-based Web-services, in leading companies' tools such as IBM and Sun Microsystems (e.g. NetBeans IDE). REST has been used in many applications such as IBM enterprise mashup solutions, Yahoo, Google Maps, Flicker, and Amazon S3.

4.2. PDES in the clouds

Cloud computing has emerged as an attractive paradigm for on-demand provisioning of computational resources to support a wide spectrum of applications [143]. Virtualization is a key technology used in cloud-enabled data centers for elastic scaling, fault tolerance, and functional isolation between applications consolidated on a shared physical platform [144]. There are three major cloud service models: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS). Among them, IaaS clouds (e.g., Amazon's EC2 [145]) allocate raw hardware resources in units of virtual machines (VMs), giving users the illusion of having their own dedicated servers to deploy whatever applications they need. The inherent elasticity of IaaS clouds allows users to carry out PDES over a cluster of VMs in a pay-as-you-go fashion without having to invest heavily in the computing facility and on-going management expenses. Nonetheless, this flexibility comes with a performance cost. As noted by Fujimoto et al. [146], cloud-based PDES suffers from extra communication delays due to the virtualization layer that lies in between the simulation program and the underlying hardware. Furthermore, the interference from the other VMs hosted on the same physical server poses a significant challenge to the parallel simulation, resulting in performance degradation especially when the simulation is synchronized by optimistic protocols. To address these issues, the authors proposed the use of the Aurora master/worker architecture, which aggregates many small messages targeting LPs on different VMs and sends the whole package as one unit to accommodate the high bandwidth, but relatively high latency interconnect in IaaS clouds. In a following paper [147], the authors went onto propose a Time Warp Straggler Message Identification Protocol (TW-SMIP) that can adjust the forward execution of LPs dynamically in an attempt to reduce cascaded rollback in cloud environments. However, the performance of TW-SMIP was evaluated on a traditional cluster with synthesized variation of background workload, instead of on a real cloud infrastructure.

In addition to the above issues, cloud-based parallel simulation also faces other difficulties. For example, individual VMs included in a virtual cluster could fail, requiring the simulation program to provide a certain degree of failure resilience at the application level. To improve load balancing and infrastructure resource utilization, VMs might be migrated to another server or suspended temporarily and resumed later by cloud service providers at runtime. More research is needed to achieve high-performance PDES in such highly dynamic and adaptive environments. For instance, in a recent effort [178] the use of simulation in the cloud using handheld devices was studied where a dedicated mobile application runs on the client side while the RISE [179] simulation server is hosted in the Cloud.

4.3. Hardware acceleration

Using special hardware to speed up PDES has been researched for several years. Earlier works employed auxiliary processors designed specifically to form a *Parallel Reduction Network* (PRN) in order to accelerate parallel global reduction as characterized by the computation of GVT and LBTS values [148]. Fujimoto et al. presented a *rollback chip* that implements state saving and rollback functions for the LPs mapped on each processor in Time Warp optimistic simulations [149]. In [150], Rosu et al. proposed an architecture for offloading communication-related functionality from the main processor to the network interface coprocessor, exposing such functionality as a *Virtual Communication Machine* (VCM) to simulation applications. This architecture enables an LP to calculate GVT based on local data only. Similarly, Noronha and Abu-Ghazaleh used programmable network interface coprocessors to detect critical messages for prompt handling. In particular, they used the technique to optimize event cancellation in Time Warp simulations [101]. By exploiting the data transfer capabilities provided by DMA (Direct Memory Access) engines in Myrinet network switches, Quaglia and Santoro implemented non-blocking state saving and other simulation operations (such as event list update) in support of optimistic PDES [151]. More recently, Lynch and Riley presented an on-chip *hardware supported global synchronization unit* that allows for efficient sharing of global state information among all processors, achieving significant performance gain in network simulations with low lookahead values [152].

As heterogeneous chip-multiprocessors continue to gain momentum in the industry [153], there is a growing interest in tapping their computation capability in parallel simulations. A number of efforts have been made towards PDES on general purpose Graphical Processing Unit (GPU), which serves as a coprocessor to the CPU and features a massive number of parallel threads executing the same code in lockstep [154]. Perumalla is among the first to propose a hybrid GPU-based PDES algorithm that performs global synchronous state updates at every discrete event point, demonstrating that the synchronous stream processing style of GPU computation can speed up discrete-event simulations [155]. To exploit the various forms of parallelism in high-fidelity network simulations, Xu and Bagrodia presented an architecture to achieve task-level parallelism on CPUs while realizing data-level parallelism on GPUs [156]. In [157], Park and Fishwick proposed an event scheduling method that divides the FEL into multiple sub-lists, each of which is processed by an individual thread on GPU. The minimum timestamp among the future events is derived by a multithreaded parallel reduction. In order to increase the number of simultaneous events that can be executed in parallel, a tolerance interval is used to group events with close timestamps at the expense of an approximation error, trading accuracy for speed. In [181] data-parallel Agent-Based Modeling implementations (ABM) on GPUs are investigated addressing the scalability issues. Moreover, in [185], the data-parallel techniques on GPUs are examined to execute very large scale direct simulation Monte Carlo (a computational method for fluid mechanics simulation), where substantial performance improvements were achieved. An interactive GPU-based evaluation of large-scale evacuation scenarios were studied in [186], providing a novel, field-based modeling technique for overcoming sever computational demands of the conventional modeling techniques.

The IBM Cell processor is another example of general purpose heterogeneous chip-multiprocessors that have been used as the building blocks in supercomputers [158]. A Cell processor combines nine independent cores based on two different instruction sets and memory subsystems on a single die, including a main two-way hardware multithreading Power Processor Element (PPE) and eight coprocessors called Synergistic Processing Elements (SPEs) [159]. The PPE is adequate for executing control-intensive code using a conventional two-level cache hierarchy, whereas each SPE is optimized to execute compute-intensive code in a SIMD (Single Instruction, Multiple Data) fashion using a small on-chip local storage. Data sharing between the cores relies on software-managed DMA transfer that requires proper memory address alignment. Since its introduction, a wide variety of scientific and multimedia applications have been successfully ported to the Cell with significant performance improvements (see, e.g., [160]). In order to realize the Cell's potential in PDES applications, Liu and Wainer proposed a technique that adopts a data-flow oriented strategy to exploit data- and event-level parallelism in a simulation, combining multi-grained parallelism and different optimizations to accelerate both memory-bound and compute-bound computational kernels [161]. Promising performance results have been obtained with large-scale discrete-event models of varied characteristics, showing that the proposed technique can achieve a significant level of scalability under different simulation workloads. Stream processing is another emerging computational model for conducting complex computations across multi-source, high-volume, unpredictable dataflows. In a recent effort [184], a platform for parallel and distributed stream processing system simulation is proposed that provides flexible modeling environment for analyzing stream processing applications.

Reconfigurable hardware, such as large Field Programmable Gate Arrays (FPGAs), exhibits the potential to deliver an order of magnitude speedup for compute-intensive kernels in scientific applications [162]. Such hardware is being integrated with general-purpose processors in next-generation reconfigurable supercomputing systems [163]. The use of FPGA has also attracted considerable attention from the PDES community. For example, Beaumont et al. presented the FPGA implementations of phase-based algorithms for global synchronization barrier and minimum timestamp computation in synchronous PDES [164]. In [165], Abu-Ghazaleh et al. explored the use of FPGA boards to accelerate Mattern's GVT algorithm in the SPEEDES simulator. A serial all-to-all connector is designed to provide low latency communication channels between the FPGAs, which keep track of the number of transit messages and broadcast new GVT updates when the transit counters become zero. In [166], Model and Herbordt proposed a FPGA-based micro-architecture for discrete-event molecular dynamics simulation, using a pipelined hardware priority queue to achieve efficient event insertion and deletion operations.

Despite these encouraging results, existing PDES techniques on heterogeneous hardware platforms are still not as mature as those developed for homogeneous multiprocessor systems. Many interesting problems such as high-performance PDES on supercomputers with diverse heterogeneous hardware configurations warrant further research, possibly through wide adoption of parallel programming standards like OpenCL [167].

5. Parallel and distributed environments

A number of environments have been developed in the past, which provide numerous services to building parallel/distributed simulation systems by supporting optimistic, conservative, or hybrid synchronization strategies. Examples of such environments are: YADDES (Yet Another Distributed Discrete Event Simulator) [129], SPEEDES (Synchronous Parallel Environment for Emulation and Discrete Event Simulation) [130], WARPED [131], HLA (High-Level Architecture) [132], WarpIV [133], Parsec (Parallel Simulation Environment for Complex Systems) [139], POSE (Parallel Object-oriented Simulation Environment) [140], JAMES II (JAva-based Multipurpose Environment for Simulation) [141], μsik [134], and Unified Framework [142].

Park and Fujimoto [138] proposed a master/worker paradigm for executing large-scale parallel discrete event simulation programs over network enabled computational resources. The paradigm adopts a client/server architecture where clients repeatedly download state vectors of logical processes and associated message data from a server (master), perform simulation computations locally at the client, and then return the results back to the server. The advantages of such approach over conventional PDES include support for execution over heterogeneous distributed computing platforms, load balancing, efficient execution on shared platforms, easy addition or removal of client machines during execution, simpler fault tolerance, and improved portability. The *Aurora Parallel and Distributed Simulation System* (Aurora) [135] is a prototype implementation of the master/worker paradigm. Several extensions and improvements to Aurora were presented later on including a scalable version for parallel discrete event simulations on desktop girds [136], an optimistic time management compliant for public-resource computing infrastructures and desktop grids [137], and a version implemented for metacomputing systems [138].

The μsik and Unified Framework protocols are well-known for their hybrid execution capabilities. The Unified Framework is a parallel simulation protocol that allows different parts of a system to be simulated using different protocols, allowing these protocols to be switched dynamically. Optimistic and conservative features can be combined and interchanged on the fly among the LPs. For example, optimistic processes can take advantage of features normally associated with the conservative LPs e.g. exploiting lookahead information and communication topology. Any of these features can be used in conjunction with a mix of conservative and optimistic LPs too. Similarly, the μsik micro-kernel provides unified system architecture for incorporating multiple types of simulation processes. Processes employ different synchronization schemes, and can dynamically alter their synchronization mechanism. μsik supports *lookahead-based* conservative and *state saving-based* optimistic execution approaches. Moreover, techniques such as *reverse computation-based* optimistic execution and *aggregation-based* event processing are also supported.

In general, hybrid synchronization techniques provide a single engine that not only supports multiple synchronization approaches, but also lowers the execution overhead. Hybrid protocols are specifically beneficial for simulating systems where it is unknown as to which synchronization scheme would perform better a priori.

6. Conclusion and future research directions

We have presented a variety of synchronization approaches used for parallel discrete event simulation, including examples of the two main classes of synchronization mechanisms, namely conservative, and optimistic. Each of the techniques was introduced, and the challenges in each class were analyzed. We showed that conservative methods offer good potential for a certain class of problems. Significant successes have been reported particularly when application-specific knowledge is applied to maximize the efficiency of the simulation mechanism. On the other hand, the optimistic methods such as Time Warp and reverse computation offer great general purpose simulation mechanisms, which, when combined with different optimization techniques, can achieve high-performance parallel simulation with low state-saving and rollback overhead. The debate of conservative versus optimistic approach remains a challenging question in the field of discrete event simulation and the decision of which technique to use depends not only on the nature of the models being simulated but also on the architecture of the computing platform in use.

With the rapid advance of emerging computing platforms, we envision that future PDES research would be oriented towards several directions, including new programming models and simulation runtime for implicit exploitation of massive parallelism offered by the underlying platform while reducing programming effort, optimization techniques for improved scalability of parallel simulation on the next generation exascale supercomputers, fault tolerance and latency hiding mechanisms for efficient parallel/distributed simulation in elastic and open virtual environments at global scale, and new techniques that address the challenges of novel heterogeneous architectures.

References

[1] R.M. Fujimoto, Parallel and Distributed Simulation Systems, Wiley, New York, 2000.
[2] R.M. Fujimoto, Parallel discrete event simulation, Communications of the ACM 33 (10) (1990) 30–53.
[3] C. Tropper, Parallel discrete-event simulation applications, Journal of Parallel and Distributed Computing 62 (2) (2002) 327–335.
[4] K.S. Perumalla, R.M. Fujimoto, Virtual time synchronization over unreliable network transport, in: Proceedings of the 15th International Workshop on Parallel and Distributed Simulation, Lake Arrowhead, CA, 2001, pp. 129–136.
[5] K.S. Perumalla, Parallel and distributed simulation: traditional techniques and recent advances, in: Proceedings of the 2006 Winter Simulation Conference, Monterey, CA, 2006, pp. 84–95.
[6] K. Venkatesh, T. Radhakrishnan, H.F. Li, Discrete Event Simulation in a Distributed System. IEEE COMPSAC, IEEE Computer Society Press, 1986.
[7] J.K. Peacock, J.W. Wong, E. Manning, Distributed simulation using a network of microcomputers, Computer Networks 3 (1979) 44–56.
[8] A. Concepcion, Mapping distributed simulators onto the hierarchical multi-bus multiprocessor architecture, in: Proceedings of 1985 SCS Multi Conference: Distributed Simulation, San Diego, CA, 1985, pp. 8–13.
[9] D.K. Baik, B.P. Zeigler, Performance evaluation of hierarchical distributed simulators, in: Proceedings of the 17th conference on Winter Simulation, San Francisco, California, United States, December 11–13, 1985, pp. 421–427.
[10] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Transactions on Computer Systems 3 (1) (1985) 63–75.
[11] F. Mattern, Efficient algorithms for distributed snapshots and global virtual time approximation, Journal of Parallel and Distributed Computing 18 (1993) 423–434.
[12] R. Garg, V.K. Garg, Y. Sabharwal, Scalable algorithms for global snapshots in distributed systems, in: Proceedings of the 20th annual international conference on Supercomputing, Cairns, Queensland, Australia, 2006.
[13] R. Garg, V.K. Garg, Y. Sabharwal, Efficient algorithms for global snapshots in large distributed systems, IEEE Transactions on Parallel and Distributed Systems 21 (5) (2010) 620–630.
[14] K.M. Chandy, J. Misra, Distributed simulation: a case study in design and verification of distributed programs, IEEE Transactions on Software Engineering 5 (5) (1979) 440–452.
[15] R.E. Bryant, Simulation of packet communication architecture computer systems, Technical report, Massachusetts Institute of Technology. Cambridge, MA, USA, 1977.
[16] A. Nketsa, N.B. Khalifa, Timed Petri nets and prediction to improve the Chandy–Misra conservative distributed simulation, Applied Mathematics and Computation 120 (1–3) (2001) 235–254.
[17] P.F. Reynolds, A shared resource algorithm for distributed simulation, in: Proceedings of the 9th Annual Computer Architecture Conference, Austin, Texas, pp. 259–266, 1982.
[18] Y.M. Teo, S.C. Tay, Efficient algorithms for conservative parallel simulation of interconnection networks, in: Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (Japan), IEEE Computer Society Press, Los Alamitos, CA, pp. 286–93. 1994.
[19] W.L. Bain, D.S. Scott, An algorithm for time synchronization in distributed discrete event simulation, in: Proceedings of SCS Multi-conference, San Diego, CA, 1988.
[20] J. Misra, Distributed discrete-event simulation, Computing Surveys 18 (1) (1986) 39–65.
[21] J.K. Peacock, J.W. Wong, E. Manning, Synchronization of distributed simulation using broadcast algorithms, Computer Networks 4 (1) (1980) 3–10.

[22] D.M. Nicol, P.F. Reynolds, Problem oriented protocol design, in: Proceedings of the 16th Conference on Winter simulation, Dallas, TX, 1984, pp. 470–474.
[23] W.K. Su, C.L. Seitz, Variants of the Chandy–Misra–Bryant distributed discrete event simulation algorithm, Proceedings of the SCS Multiconference on Distributed Simulation 21 (1) (1989) 38–43.
[24] N.J. Davis, D.L. Mannix, W.H. Shaw, T.C. Hartrum, Distributed discrete-event simulation using null message algorithms on hypercube architectures, Journal of Parallel and Distributed Computing 8 (4) (April 1990) 349–357.
[25] W. Cai, S.J. Turner, An algorithm for distributed discrete-event simulation – the 'Carrier Null Message' approach, Proceedings of the SCS Multiconference on Distributed Simulation, SCS 22 (1) (1990) 3–8.
[26] K.R. Wood, S.J. Turner, A generalized carrier-null method for conservative parallel simulation, in: Proceedings of the Eighth Workshop on Parallel and Distributed Simulation, Edinburgh, Scotland, United Kingdom, 1994, pp. 50–57.
[27] B.R. Preiss, W.M. Loucks, J.D. MacIntyre, J.A. Field, Null message cancellation in conservative distributed simulation, in: Proceedings of the 1991 Workshop on Parallel and Distributed Simulation, Institute of Electrical and Electronics Engineers/Society for Computer Simulation, Anaheim, CA, January 1991, pp. 33–38.
[28] J. Porras, V. Hara, J. Jarju, J. Ikonen, Improving the performance of the Chandy–Misra parallel simulation algorithm in a distributed workstation environment, in: Proceedings of the SCSC'97, Arlington, VA, 1997, pp. 657–662.
[29] E. Naroska, U. Schwiegelshohn, A new scheduling method for parallel discrete-event simulation, Proceedings of the Second International Euro-Par Conference on Parallel Processing 2 (1) (1996) 582–593.
[30] Z. Xiao, B. Unger, R. Simmonds, J. Cleary, Scheduling critical channels in conservative parallel discrete event simulation, in: Proceedings of the 13th International Workshop on Parallel and Distributed Simulation, Atlanta, GA, 1999, pp. 20–28.
[31] R. Simmonds, C. Kiddle, B. Unger, Addressing blocking and scalability in critical channel traversing, in: Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation, Washington, DC, May 12–15, 2002.
[32] A. Boukerche, S. Das, Reducing null messages overhead through load balancing in conservative distributed simulation systems, Journal of Parallel and Distributed Computing 64 (3) (2004) 330–344.
[33] S. Rizvi, K.M. Elleithy, A. Riasat, A new mathematical model for optimizing the performance of parallel and discrete event simulation systems, in: 11th Communications and Networking Simulation Symposium (CNS'08), Part of the 2008 Spring Simulation Multiconference (SpringSim'08), Crown Plaza, Ottawa, Ontario, 2008.
[34] B. Thomas, S. Rizvi, K.M. Elleithy, Reducing null messages using grouping and status retrieval for a conservative discrete-event simulation system, in: Proceedings of the 2009 Spring Simulation Multiconference (SpringSim'09), San Diego, CA, 2009.
[35] E. Deelman, R. Bagrodia, R. Sakellariou, V. Adve, Improving lookahead in parallel discrete event simulations of large-scale applications using compiler analysis, in: Proceedings of the Fifteenth Workshop on Parallel and Distributed Simulation, Lake Arrowhead, California, United States, 2001, pp. 5–13.
[36] J. Liu, K. Tan, D. Nicol, Lock-free scheduling of logical processes in parallel discrete-event simulation, in: Proceedings of the Fifteenth Workshop on Parallel and Distributed Simulation, Lake Arrowhead, California, United States, 2001, pp. 22–31.
[37] V. Solcany, J. Safarik, The lookahead in a user-transparent conservative parallel simulator, in: Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation, Washington, DC, May 12–15, 2002.
[38] M.K. Chung, C.M. Kyung, Improving lookahead in parallel multiprocessor simulation using dynamic execution path prediction, in: Proceedings of PADS'06, Singapore, May 24–26, 2006.
[39] A. Park, R.M. Fujimoto, K.S. Perumalla, Conservative synchronization of large-scale network simulations, in: Proceedings of the 18th International Workshop on Parallel and Distributed Simulation, Kufstein, Austria, pp. 153–161, 2004.
[40] R.L. Bagrodia, M. Takai, Performance evaluation of conservative algorithms in parallel simulation languages, IEEE Transactions on Parallel and Distributed Systems 11 (4) (2000) 395–411.
[41] H.Y. Song, R.A. Meyer, R. Bagrodia, An empirical study of conservative scheduling, in: Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'00), Bologna, Italy, 2000.
[42] S. Jafer, G. Wainer, Global lookahead management (GLM) protocol for conservative DEVS simulation, in: DS-RT '10 Proceedings of the 2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications, Fairfax, Virginia, USA, 17–20 October, 2010, pp. 141–148.
[43] C. Kawabata, R. Santana, M. Santana, S. Bruschi, K. Castelo Branco, Performance evaluation of a CMB protocol, in: Proceedings of the 38th Conference on Winter Simulation, IEEE Computer Society, Los Alamitos, CA, December 2006, pp. 1012–1019.
[44] K. Chandy, J. Misra, Asynchronous distributed simulation via a sequence of parallel computations, Communications of the ACM 24 (2) (1981) 198–205.
[45] O. Berry, D. Jefferson, Critical path analysis of distributed simulation, in: Proceedings of 1985 SCS Multiconference on Distributed, Simulation, January 1985, pp. 57–60.
[46] S. Srinivasan, P. Reynolds, On critical path analysis of parallel discrete event simulations, Technical report no. CS-93-29, 1993.
[47] S. Lin, X. Cheng, J. Lv, State causality analysis of conservative parallel network simulation, in: Proceedings of 41st Annual Simulation Symposium (ANSS-41 2008), Ottawa, Canada, April 14–16, 2008, pp. 251–260.
[48] B. Groselj, C. Tropper, The time of next event algorithm, in: Proceedings of the 1988 Distributed Simulation Conference, SCS Simulation Series 19(3), pp. 25–29, 1988.
[49] A. Boukerche, C. Tropper, SGTNE: semi-global time of the next event algorithm, in: Proceedings of the Ninth Workshop on Parallel and Distributed Simulation, Lake Placid, New York, United States, June 13–16, 1995, pp. 68–77.
[50] R.M. Fujimoto, Distributed simulation systems, in: Proceedings of the 2003 Winter Simulation Conference, New Orleans, LA, 2003, pp. 124–134.
[51] R. Curry, C. Kiddle, R. Simmonds, B. Unger, Sequential performance of asynchronous conservative PDES algorithms, in: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, June 01–03, 2005, pp. 217–226.
[52] B.D. Lubachevsky, Efficient distributed event-driven simulations of multiple-loop networks, Communications of the ACM 32 (January 1989) 111–123.
[53] L.M. Sokol, J.B. Weissman, P.A. Mutchler, MTW: an empirical performance study, in: Proceedings of the 1991 Winter Simulation Conference, Phoenix, AZ, 1991, pp. 557–563.
[54] R. Ayani, H. Rajaei, Parallel simulation using conservative time windows, in: Proceedings of the 24th Conference on Winter Simulation, Arlington, Virginia, United States, December 13–16, 1992, pp. 709–717.
[55] J. Lemeire, E. Dirkx, Lookahead accumulation in conservative parallel discrete event simulation, in: Proceedings of the of the High Performance Computing and Simulation (HPCS) Conference, Magdeburg, Germany, June 13–16, 2004.
[56] S. Lin, X. Cheng, J. Lv, Micro-synchronization in conservative parallel network simulation, in: ACMlIEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation, Lake Placid, NY, USA, June 2009.
[57] H. Rajaei, Local time warp: an implementation and performance analysis, in: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, San Diego, CA, 2007, pp. 163–170.
[58] R. Ayani, A parallel simulation scheme based on the distance between objects, Proceedings of the SCS Multiconference on Distributed Simulation 21 (2) (1989) 113–118.
[59] B.D. Lubachevsky, Bounded lag distributed discrete event simulation, in: Proceedings of the 1988 Distributed Simulation Conference, SCS Simulation Series 19(3), 1988, pp. 183–191.
[60] K.M. Chandy, R. Sherman, The conditional event approach to distributed simulation, in: Proceedings of SCS Multiconference on Distributed, Simulation, 21, 1989, pp. 93–99.
[61] D.M. Nicol, The cost of conservative synchronization in parallel discrete event simulations, Tech. Rep. 90-20, ICASE, June 1989.

[62] D. Nicol, J. Liu, Composite synchronization in parallel discrete-event simulation, IEEE Transactions on Parallel and Distributed Systems 13 (5) (May 2002) 433–446.

[63] A. Boukerche, Conservative circuit simulation on multiprocessor machines, in: Proceedings of the 7th International Conference on High Performance Computing, Bangalore, India, LNCS 1970, 2000, pp. 415–424.

[64] J. Liu, D.M. Nicol, Lookahead revisited in wireless network simulations, in: Proceedings of the 16th International Workshop on Parallel and Distributed Simulation, Washington, DC, pp. 79–88, 2002.

[65] D.R. Jefferson, Virtual time, ACM Transactions on Programming Languages and Systems 7 (3) (1985) 405–425.

[66] C.D. Carothers, K.S. Perumalla, On deciding between conservative and optimistic approaches on massively parallel platforms, in: Proceedings of the 2010 Winter Simulation Conference, Baltimore, MD, 2010, pp. 678–687.

[67] R. Beraldi, L. Nigro, Performance of a time warp based simulator of large scale PCS networks, Simulation Practice and Theory 6 (2) (1998) 149–163.

[68] F. Wieland, L. Hawley, A. Feinberg, M.D. Loreto, L. Blume, J. Ruffles, P. Reiher, B. Beckman, P. Hontalas, S. Bellenot, D. Jefferson, The performance of a distributed combat simulation with the time warp operating system, in: Concurrency: Practice and Experience, 1(1), pp. 35–50, 2006.

[69] M.T. Presley, P.L. Reiher, S.F. Bellenot, A time warp implementation of sharks world, in: Proceedings of the 1990 Winter Simulation Conference, New Orleans, LA, pp. 199–203, 1990.

[70] G. Yaun, C.D. Carothers, S. Adali, D. Spooner, Optimistic parallel simulation of a large-scale view storage system, Future Generation Computer Systems 19 (4) (2003) 479–492.

[71] D.R. Jefferson, Virtual time II: storage management in distributed simulation, in: Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, Quebec, Canada, pp. 75–89, 1990.

[72] Y.B. Lin, B.R. Preiss, Optimal memory management for time warp parallel simulation, ACM Transactions on Modeling and Computer Simulation 1 (4) (1991) 283–307.

[73] B.R. Preiss, W.M. Loucks, Memory management techniques for time warp on a distributed memory machine, in: Proceedings of the 9th International Workshop on Parallel and Distributed Simulation, Lake Placid, NY, pp. 30–39, 1995.

[74] C.H. Young, N.B. Abu-Ghazaleh, P.A. Wilsey, OFC: a distributed fossil collection algorithm for time warp, in: Proceedings of the 12th International Symposium on Distributed Computing, Andros, Greece, LNCS 1499, pp. 408–418, 1998.

[75] C.H. Young, R. Radhakrishnan, P.A. Wilsey, Optimism: not just for event execution anymore, in: Proceedings of the 13th International Workshop on Principles of Advanced and Distributed Simulation, Atlanta, GA, pp. 136–143, 1999.

[76] M. Chetlur, P.A. Wilsey, Causality information and fossil collection in time warp simulations, in: Proceedings of the 2006 Winter Simulation Conference, Monterey, CA, pp. 987–994, 2006.

[77] V.Y. Vee, W.J. Hsu, Pal: a new fossil collector for time warp, in: Proceedings of the 16th International Workshop on Parallel and Distributed Simulation, Washington, DC, pp. 35–42, 2002.

[78] J. Fleischmann, P.A. Wilsey, Comparative analysis of periodic state saving techniques in time warp simulations, in: Proceedings of the 9th International Workshop on Parallel and Distributed Simulation, Lake Placid, NY, pp. 50–58, 1995.

[79] H. Bauer, C. Sporrer, Reducing rollback overhead in time-warp based distributed simulation with optimized incremental state saving, in: Proceedings of the 26th Annual Simulation Symposium, Arlington, VA, pp. 12–20, 1993.

[80] S.C. Tay, Y.M. Teo, Probabilistic checkpointing in time warp parallel simulation, in: Proceedings of the 8th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, San Francisco, CA, pp. 366–373, 2000.

[81] C.D. Carothers, K.S. Perumalla, R.M. Fujimoto, Efficient optimistic parallel simulations using reverse computation, ACM Transactions on Modeling and Computer Simulation 9 (3) (1999) 224–253.

[82] C.D. Carothers, D. Bauer, S. Pearce, ROSS: a high-performance, low-memory, modular time warp system, Journal of Parallel and Distributed Computing 62 (11) (2002) 1648–1669.

[83] Y. Tang, K.S. Perumalla, R.M. Fujimoto, H. Karimabadi, J. Driscoll, Y. Omelchenko, Optimistic simulations of physical systems using reverse computation, Simulation 82 (1) (2006) 61–73.

[84] D.W. Bauer, E.H. Page, An approach for incorporating rollback through perfectly reversible computation in a stream simulator, in: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, San Diego, CA, pp. 171–178, 2007.

[85] L. Li, C. Tropper, Event reconstruction in time warp, in: Proceedings of the 18th International Workshop on Principles of Advanced and Distributed Simulation, Kufstein, Austria, pp. 37–44, 2004.

[86] Q. Liu, G. Wainer, Parallel environment for DEVS and cell-DEVS models, Simulation 83 (6) (2007) 449–471.

[87] Q. Liu, G. Wainer, Lightweight time warp – a novel protocol for parallel optimistic simulation of large-scale DEVS and cell-DEVS models, in: Proceedings of the 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, Vancouver, Canada, pp. 131–138, 2008.

[88] Q. Liu, G. Wainer, A performance evaluation of the lightweight time warp protocol in optimistic parallel simulation of DEVS-based environmental models, in: Proceedings of the 23rd International Workshop on Principles of Advanced and Distributed Simulation, Lake Placid, NY, pp. 27–34, 2009.

[89] J.S. Steinman, Breathing time warp, ACM SIGSIM Simulation Digest 23 (1) (1993) 109–118.

[90] S.C. Tay, Y.M. Teo, Performance optimization of throttled time-warp simulation, in: Proceedings of the 34th Annual Simulation Symposium, Seattle, WA, pp. 211–218, 2001.

[91] S. Srinivasan, P.F. Reynolds, Elastic time, ACM Transactions on Modeling and Computer Simulation 8 (2) (1998) 103–139.

[92] R. Suppi, F. Cores, E. Luque, An efficient method for improving large optimistic PDES, in: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, pp. 351-357, 2000.

[93] V. Sachdev, M. Hybinette, E. Kraemer, Controlling over-optimism in time-warp via CPU-based flow control, in: Proceedings of the 2004 Winter Simulation Conference, Washington, DC, pp. 402–410, 2004.

[94] J. Wang, C. Tropper, Optimizing time warp simulation with reinforcement learning techniques, in: Proceedings of the 2007 Winter Simulation Conference, Washington, DC, pp. 577–584, 2007.

[95] S.R. Das, Adaptive protocols for parallel discrete event simulation, in: Proceedings of the 1996 Winter Simulation Conference, Coronado, CA, pp. 186–193, 1996.

[96] K.S. Panesar, R.M. Fujimoto, Adaptive flow control in time warp, in: Proceedings of the 11th International Workshop on Parallel and Distributed Simulation, Lockenhaus, Austria, pp. 108–115, 1997.

[97] A. Gafni, Rollback mechanisms for optimistic distributed simulation systems, in: Proceedings of the SCS Multiconference on Distributed Simulation, San Diego, CA, pp. 61–67, 1988.

[98] Y.B. Lin, E.D. Lazowska, A study of time warp rollback mechanisms, ACM Transactions on Modeling and Computer Simulation 1 (1) (1991) 51–72.

[99] H.M. Soliman, Throttled lazy cancellation in time warp parallel simulation, Simulation 84 (2–3) (2008) 149–160.

[100] V. Madisetti, J. Walrand, D. Messerschmitt, WOLF: a rollback algorithm for optimistic distributed simulation systems, in: Proceedings of the 1988 Winter Simulation Conference, San Diego, CA, pp. 296–305, 1988.

[101] R. Noronha, N.B. Abu-Ghazaleh, Early cancellation: an active NIC optimization for time-warp, in: Proceedings of the 16th International Workshop on Parallel and Distributed Simulation, Washington, DC, pp. 43–50, 2002.

[102] M. Chetlur, P.A. Wilsey, Causality representation and cancellation mechanisms in time warp simulations, in: Proceedings of the 15th International Workshop on Parallel and Distributed Simulation, Lake Arrowhead, CA, pp. 165–172, 2001.

[103] M. Chetlur, P.A. Wilsey, Causality information and proactive cancellation mechanisms, Concurrency and Computation: Practice and Experience 21 (11) (2009) 1483–2503.

[104] Y. Zeng, W. Cai, S.J. Turner, Batch based cancellation: a rollback optimal cancellation scheme in time warp simulations, in: Proceedings of the 18th International Workshop on Principles of Advanced and Distributed Simulation, Kufstein, Austria, pp. 78–86, 2004.

[105] A. Boukerche, A. Mikkler, A. Fabri, Resource control for large-scale distributed simulation system over loosely coupled domains, Journal of Parallel and Distributed Computing 65 (10) (2005) 1171–1189.
[106] H.M. Soliman, A.S. Elmaghraby, An efficient clustered adaptive-risk technique for distributed simulation, in: Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing, Syracuse, NY, pp. 383–391, 1996.
[107] H. Avril, C. Tropper, Clustered time warp and logic simulation, in: Proceedings of the 9th International Workshop on Parallel and Distributed Simulation, Lake Placid, NY, pp. 112–119, 1995.
[108] H. Avril, C. Tropper, On rolling back and checkpointing in time warp, IEEE Transactions on Parallel and Distributed Systems 12 (11) (2001) 1105–1121.
[109] R. Ronngren, R. Ayani, R.M. Fujimoto, S.R. Das, Efficient implementation of event sets in time warp, ACM SIGSIM Simulation Digest 23 (1) (1993) 101–108.
[110] S.D. Nikolopoulos, R. MacLeod, An experimental analysis of event set algorithms for discrete event simulation, Microprocessing and Microprogramming 36 (2) (1993) 71–81.
[111] J.S. Steinman, Discrete-event simulation and the event horizon. Part 2: event list management, in: Proceedings of the 10th International Workshop on Parallel and Distributed Simulation, Philadelphia, PA, pp. 170–178, 1996.
[112] D.W. Jones, An empirical comparison of priority-queue and event-set implementations, Communications of the ACM 29 (4) (1986) 300–311.
[113] R. Ronngren, J. Riboe, R. Ayani, Lazy queue: an efficient implementation of the pending-event set, in: Proceedings of the 24th Annual Simulation Symposium, New Orleans, LA, pp. 194–204, 1991.
[114] W.T. Tang, R.S.M. Goh, L.J. Thng, Ladder queue: an O(1) priority queue structure for large-scale discrete event simulation, ACM Transactions on Modeling and Computer Simulation 15 (3) (2005) 175–204.
[115] R. Brown, Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem, Communications of the ACM 31 (10) (1988) 1220–1227.
[116] F. Wieland, The threshold of event simultaneity, in: Proceedings of the 11th International Workshop on Parallel and Distributed Simulation, Lockenhaus, Austria, pp. 56–59, 1997.
[117] V. Jha, R. Bagrodia, Simultaneous events and lookahead in simulation protocols, ACM Transactions on Modeling and Computer Simulation 10 (3) (2000) 241–267.
[118] H. Mehl, A deterministic tie-breaking scheme for sequential and distributed simulation, in: Proceedings of the SCS Multiconference on Parallel and Distributed Simulation, Newport Beach, CA, 1992.
[119] P. Peschlow, P. Martini, Efficient analysis of simultaneous events in distributed simulation, in: Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications, Chania, Greece, pp. 244–251, 2007.
[120] X. Wang, S.J. Turner, S.J.E. Taylor, COTS simulation package (CSP) interoperability – a solution to synchronous entity passing, in: Proceedings of the 20th International Workshop on Principles of Advanced and Distributed Simulation, Singapore, pp. 201–210, 2006.
[121] M.H. Willebeek-Lemair, A.P. Reeves, Strategies for dynamic load balancing on highly parallel computers, IEEE Transactions on Parallel and Distributed Systems 4 (9) (1993) 979–993.
[122] P.L. Reiher, D.R. Jefferson, Virtual time based dynamic load management in the time warp operating system, in: Proceedings of the SCS Multiconference on Parallel and Distributed Simulation, San Diego, CA, pp. 103–111, 1990.
[123] A. Boukerche, S.K. Das, Dynamic load balancing strategies for conservative parallel simulations, in: Proceedings of the 11th International Workshop on Parallel and Distributed Simulation, Lockenhaus, Austria, pp. 20–28, 1997.
[124] L.F. Wilson, W. Shen, Experiments in load migration and dynamic load balancing in SPEEDS, in: Proceedings of the 1998 Winter Simulation Conference, Washington, DC, pp. 483–490, 1998.
[125] P. Peschlow, T. Honecker, P. Martini, A flexible dynamic partitioning algorithm for optimistic distributed simulation, in: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, San Diego, CA, pp. 219–228, 2007.
[126] Y. Artsy, R. Finkel, Designing a process migration facility: the charlotte experience, IEEE Computer 22 (9) (1989) 47–56.
[127] J.S. Steinman, Interactive SPEEDES, in: Proceedings of the 24th Annual Simulation Symposium, New Orleans, LA, pp. 149–158, 1991.
[128] H. Avril, C. Tropper, The dynamic load balancing of clustered time warp for logic simulation, in: Proceedings of the 10th International Workshop on Parallel and Distributed Simulation, Philadelphia, PA, pp. 20–27, 1996.
[129] B.R. Preiss, The Yaddes distributed discrete event simulation specification language and execution environments, in: Proceedings SCS Eastern MultiConference — Distributed Simulation, Society for Computer Simulation, vol. 21, no. 2, pp. 139–144, 1989.
[130] J.S. Steinman, SPEEDES: a unified approach to parallel simulation, in: Proceedings of 6th Workshop on Parallel and Distributed Simulation, SCS Simulation Series 24(3), Newport Beach, CA, January 20–22, pp. 75–84.
[131] R. Radhakrishnan, D.E. Martin, M. Chetlur, D.M. Rao, P.A. Wilsey, An object-oriented time warp simulation kernel, in: Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments, ISCOPE'98, Lecture Notes in Computer Science 1505(1), Springer, Berlin, December 1998, pp. 13–23.
[132] sIEEE std 1516.2-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification. Institute of Electrical and Electronic Engineers, New York, NY, 2001.
[133] J. Steinman, The WarpIV simulation kernel, in: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, Washington, DC, USA, 2005, pp. 161–170.
[134] K.S. Perumalla, μsik – A micro-kernel for parallel/distributed simulation systems, in: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, Washington, DC, USA, 2005, pp. 59–68.
[135] A. Park, R.M. Fujimoto, Aurora: an approach to high throughput parallel simulation, in: Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation, May 24–26, 2006, pp. 3–10.
[136] A. Park, R.M. Fujimoto, A scalable framework for parallel discrete event simulations on desktop grids, in: 8th IEEE/ACM International Conference on Grid Computing, Austin, TX, 2007, pp. 185–192.
[137] A. Park, R.M. Fujimoto, Optimistic parallel simulation over public resource-computing infrastructures and desktop grids, in: Proceedings of 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, Vancouver, BC, Canada, 2008.
[138] A. Park, R.M. Fujimoto, Efficient master/worker parallel discrete event simulation, in: Proceedings of the 23rd Workshop on Principles of Advanced and Distributed Simulation, IEEE, Picataway, NJ, 2009, pp. 145–152.
[139] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, H.Y. Song, Parsec: a parallel simulation environment for complex systems, Computer 31 (10) (1998) 77–85.
[140] T.L. Wilmarth, L.V. Kale, POSE: getting over grainsize in parallel discrete event simulation, in: Proceedings of the 2004 International Conference on Parallel Processing, Montreal, Canada, pp. 12–19. 2004.
[141] B. Wang, Y. Yao, J. Himmelspach, R. Ewald, A.M. Uhrmacher, Experimental analysis of logical process simulation algorithms in JAMES II, in: Proceedings of the 2009 Winter Simulation Conference, Austin, TX, pp. 1167–1179, 2009.
[142] V. Jha, R. Bagrodia, A unified framework for conservative and optimistic distributed simulation, in: Proceedings of the Eighth Workshop on Parallel and Distributed Simulation, Edinburgh, Scotland, United Kingdom, July 06–08, 1994, pp. 12–19.
[143] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the clouds: a berkeley view of cloud computing, in: Technical Report No. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, CA, 2009.
[144] L.M. Vaquero, L.R. Merino, J. Caceres, M. Lindner, A break in the clouds: towards a cloud definition, ACM SIGCOMM Computer Communication Review 39 (1) (2009) 50–55.
[145] Amazon Corporation, Elastic Compute Cloud, 2012. <http://www.aws.amazon.com/ec2/> (accessed February 14).

[146] R.M. Fujimoto, A.W. Malik, A.J. Park, Parallel and distributed simulation in the cloud, SCS M&S Magazine (3), 2010.
[147] A.W. Malik, A.J. Park, R.M. Fujimoto, An optimistic parallel simulation protocol for cloud computing environments, SCS M&S Magazine (4), 2010.
[148] P.F. Reynolds, C.M. Pancerella, S. Srinivasan, Design and performance analysis of hardware support for parallel simulations, Journal of Parallel and Distributed Computing 18 (1) (1993) 435–453.
[149] R.M. Fujimoto, J.-J. Tsai, G.C. Gopalakrishnan, Design and evaluation of the rollback chip: special purpose hardware for time warp, IEEE Transactions on Computers 41 (1) (1992) 68–82.
[150] M.-C., Rosu K. Schwan, R.M. Fujimoto, Supporting parallel applications on clusters of workstations: the intelligent network interface approach, in: Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, Portland, OR, pp. 159–168, 1997.
[151] A. Santoro, F. Quaglia, Multiprogrammed non-blocking checkpoints in support of optimistic simulation on myrinet clusters, Journal of Systems Architecture 53 (9) (2007) 659–676.
[152] E.W. Lynch, G.F. Riley, Hardware supported time synchronization in multi-core architectures, in: Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed, Simulation (PADS'09), 2009, pp. 88–94.
[153] R. Kumar, D.M. Tullsen, N.P. Jouppi, P. Ranganathan, Heterogeneous chip multiprocessors, Computer 38 (11) (2005) 32–38.
[154] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, T.J. Purcell, A survey of general-purpose computation on graphics hardware, Computer Graphics Forum 26 (1) (2007) 80–113.
[155] K.S. Perumalla, Discrete-event execution alternatives on general purpose graphical processing units (GPGPUs), in: Proceedings of the 20th IEEE Workshop on Principles of Advanced and Distributed Simulation, Singapore, pp. 74–81, 2006.
[156] Z. Xu, R. Bagrodia, GPU-accelerated evaluation platform for high fidelity network modeling, in: Proceedings of the 21st IEEE Workshop on Principles of Advanced and Distributed Simulation, San Diego, CA, pp. 131–140, 2007.
[157] H. Park, P.A. Fishwick, A GPU-based application framework supporting fast discrete-event simulation, Simulation: International Transactions of the Society for Modeling and Simulation 86 (10) (2010) 613–628.
[158] K.J. Barker, K. Davis, A. Hoisie, D.J. Kerbyson, M. Lang, S. Pakin, J.C. Sancho, Entering the Petaflop era: the architecture and performance of roadrunner, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Austin, TX, USA, 2008.
[159] T. Chen, R. Raghavan, J.N. Dale, E. Iwata, Cell broadband engine architecture and its first implementation – a performance view, IBM Journal of Research and Development 51 (5) (2007) 559–572.
[160] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, K. Yelick, The potential of the cell processor for scientific computing, in: Proceedings of the 3rd Conference on Computing Frontiers, Ischia, Italy, pp. 9–20, 2006.
[161] Q. Liu, G. Wainer, Multicore acceleration of discrete event system specification systems, Simulation 88 (7) (2012) 801–831.
[162] M.C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, D. DiSabello, Achieving high performance with FPGA-based computing, IEEE Computer 40 (3) (2007) 50–57.
[163] K.D. Underwood, K.S. Hemmert, C.D. Ulmer, From silicon to science. The long road to production reconfigurable supercomputing, in: ACM transactions on reconfigurable technology and systems, vol. 2, no. 4, Article 26, 2009.
[164] C. Beaumont, P. Boronat, J. Champeau, J.-M. Filloque, B. Pottier, Reconfigurable technology: an innovative solution for parallel discrete event simulation support, in: Proceedings of the 8th IEEE Workshop on Parallel and Distributed Simulation, Edinburgh, UK, pp. 160–163, 1994.
[165] N. Abu-Ghazaleh, R. Linderman, R. Hillman, J. Hanna, Exploiting HHPC for parallel discrete event simulation, in: Proceedings of the 31st Annual International Symposium on Computer Architecture, Williamsburg, VA, pp. 250–253, 2004.
[166] J. Model, M.C. Herbordt, Discrete event simulation of molecular dynamics with configurable logic, in: Proceedings of the 2007 International Conference on Field Programmable Logic and Applications, Amsterdam, Netherlands, pp. 151–158, 2007.
[167] Khronos OpenCL Working Group, The OpenCL Specification (Version 1.2). <http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf> (accessed February 14).
[168] G. Wainer, K. Al-Zoubi, An introduction to distributed simulation, in: Catherine Banks, John Sokolowski (Eds.), Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains, Wiley, New Jersey, 2010 (Chapter 11 in book).
[169] S. Strassburger, T. Schulze, R. Fujimoto, Future trends in distributed simulation and distributed virtual environments: results of a peer study, in: Proceedings of Winter Simulation Conference (WSC 2008), Miami, FL, USA, 2008.
[170] G. Wainer, K. Al-Zoubi, S. Mittal, J.L. Risco, H. Sarjoughian, B.P. Zeigler, DEVS standardization: foundations and trends, in: G. Wainer, P. Mosterman (Eds.), Discrete-Event Modeling and Simulation: Theory and Applications, CRC Press, Taylor and Francis, 2010 (Chapter 15).
[171] L. Richardson, S. Ruby, RESTful Web Services, first ed., O'Reilly Media, Inc., Sebastopol, California, 2007.
[172] IEEE: Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT) Specification. Technical Report 1516.2, IEEE, 2000.
[173] C. Boer, A. Bruin, A. Verbraeck, A survey on distributed simulation in industry, Journal of Simulation 1 (3) (2009) 3–16.
[174] B. Khargharia, S. Hariri, M. Parashar, L. Ntaimo, B. Kim, vGrid: a framework for building autonomic applications, In: Proceedings of 1st International Workshop on Heterogeneous and Adaptive Computing– Challenges of Large Applications in Distributed Environments (CLADE 2003), Seattle, WA, USA, IEEE Computer Society Press, Los Alamitos, pp. 19–26, 2003.
[175] B.J. Wilson, JXTA. New Riders Publishing, 2002.
[176] IBM Mashup Center, 2009. <http://www-01.ibm.com/software/info/mashup-center/> (accessed June 2009).
[177] G. Wainer, S. Jafer, Q. Liu, Advanced parallel simulation of DEVS models in CD++, in: G. Wainer, P. Mosterman (Eds.), Discrete-Event Modeling and Simulation: Theory and Applications, Taylor and Francis, London, 2010.
[178] E. Mancini, G. Wainer, K. Al-Zoubi, O. Dalle, Simulation in the cloud using handheld devices, in: MSGC'12 Workshop, CCGRID 2012, Ottawa, ON, 2012.
[179] K. Al-Zoubi, G. Wainer, RISE: REST-ing heterogeneous simulations interoperability, in: Proceedings of the Winter Simulation Conference, Baltimore, MD, 2010.
[180] J. Liu, Y. Li, Parallel hybrid network traffic models, Simulation 85 (4) (2009) 271–286.
[181] S. Samuel Alberts, M. Keenan, R. D'Souza, Data-parallel techniques for simulating a mega-scale agent-based model of systemic inflammatory response syndrome on graphics processing units, Simulation 88 (8) (2012) 895–907 (first published on November 17, 2011).
[182] R. Moreno, A. Robles-Gómez, A. Bermúdez, R. Casado, SensGrid: modeling and simulation for wireless sensor grids, Simulation 88 (8) (2012) 972–987 (first published on January 19, 2012).
[183] K. Perumalla, S. Seal, Discrete event modeling and massively parallel execution of epidemic outbreak phenomena, Simulation 88 (8) (2012) 768–783 (first published on July 25, 2011).
[184] A. Park, C. Li, R. Nair, N. Ohba, U. Shvardon, A. Zaks, E. Schenfeld, Towards flexible exascale stream processing system simulation, Simulation 88 (7) (2012) 832–851 (first published on August 9, 2011).
[185] D. Gladkov, J. Tapia, S. Alberts, R. D'souza, Graphics processing unit based direct simulation Monte Carlo, Simulation 8 (6) (2012) 680–693 (first published on September 26, 2011).
[186] K. Perumalla, B. Aaby, S. Yoginath, S. Seal, Interactive, graphical processing unit-based evaluation of evacuation scenarios at the state scale, Simulation 88 (6) (2012) 746–761 (first published on October 23, 2011).
[187] G. Wainer, E. Glinsky, M. Gutierrez-Alcaraz, Studying performance of DEVS modeling and simulation environments using the DEVStone benchmark, Simulation 87 (7) (2011) 555–580 (first published on January 20, 2011).
[188] R. Buyya, M. Murshed, GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Journal of Concurrency and Computation: Practice Experience (CCPE) 14 (13) (2002) 1175–1220.