2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT) | 978-1-6654-9799-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/DS-RT55542.2022.9932067

Comparing Speculative Synchronization Algorithms for Continuous-Time Agent-Based Simulations

Philipp Andelfinger*, Andrea Piccione†, Alessandro Pellegrini‡, and Adelinde Uhrmacher*
*Institute for Visual and Analytic Computing. University of Rostock. Rostock, Germany.
†Sapienza, University of Rome. Rome, Italy.
‡University of Rome Tor Vergata. Rome, Italy.

***Abstract*—Continuous-time agent-based models often represent tightly-coupled systems in which an agent's state transitions occur in close interaction with neighboring agents. Without artificial discretization, the potential for near-instantaneous propagation of effects across the model presents a challenge to parallelizing their execution. Although existing algorithms can tackle the largely unpredictable nature of such simulations through speculative execution, they are subject to trade-offs concerning the degree of optimism, the probability and cost of rollbacks, and the exploitation of locality. This paper is aimed at understanding the suitability of asynchronous and synchronous parallel simulation algorithms when executing continuous-time agent-based models with rate-driven stochastic transitions. We present extensive measurement results comparing optimized implementations under various configurations of a parametrizable simulation model of the epidemic spread of disease. Our results show that the amount of locality in the agent interactions is the decisive factor for the relative performance of the approaches. Based on profiling results, we identify remaining hurdles for higher simulation performance with the two classes of algorithms and outline potential refinements.**

I. Introduction

Many real-world phenomena are naturally described as memoryless stochastic processes in which transitions occur with exponentially distributed delays in continuous time. In Gillespie's Stochastic Simulation Algorithm (SSA), chemical reactions occur after delays drawn from exponential distributions, whose rates depend on physical constants and the variable number of potential reaction partners. Reactions compete in a *stochastic race* based on their randomly drawn delays to determine the next occurring reaction. Several variants of SSA exist, of which the Next Reaction Method (NRM) [1] follows a familiar discrete-event mode of execution. Potential reactions are scheduled in the form of timestamped events. The simulation proceeds by iteratively selecting and actuating the reaction with the smallest timestamp, during which dependent reactions are rescheduled to account for changes in their rates. In recent years, this approach found its way into individual-based simulations in domains such as epidemics [2] and demographics [3]. Here, stochastic races decide which potential agents' transition takes place next. At each transition, agents may access arbitrary portions of the simulation state as well as change other agents' transition rates.

Applications of SSA in the context of agent-based modeling inherit the algorithms' key properties. On the one hand, SSA allows for an exact, albeit stochastic, execution of a given model specification. On the other hand, SSA-driven simulations are expensive to scale to large models. Firstly, the selection of the next transition from a large number of potential transitions incurs substantial overhead. Secondly, the dynamic updates of dependent rates introduce the potential for tight coupling along the sequence of transitions.

Discrete-event simulations of large systems are commonly accelerated using methods from parallel and distributed simulation [4], in which the computational load is distributed to a set of processors interconnected via shared memory or a network. A variety of synchronization algorithms exist to account for the dependencies among model portions referred to as logical processes (LPs), executed by separate processors. Conservative synchronization algorithms guarantee that the temporal ordering of transitions is maintained throughout the entire execution of a simulation run. To rule out ordering violations, conservative algorithms rely on *lookahead*, which is the ability for processors to predict their effects on other processors into some quantity of the simulated future. In contrast, optimistic (also referred to as speculative) algorithms detect ordering violations, after which the simulation is rolled back to a previous state.

The stochastic nature and dynamic rate updates of SSA-driven agent-based simulations and the resulting difficulty in predicting the times of future transitions render SSA-driven simulations largely unfit for conservative synchronization. However, optimistic algorithms also face substantial challenges. Time Warp [5], arguably the most widely known optimistic algorithm, allows processors to advance in simulation time asynchronously. Each transition may involve instantaneous read/write access to agent states located on separate processors. Correctly serving such accesses couples the source and destination agents' progress in simulation time, leading to rollbacks or idle times. Moreover, Time Warp's assumption of a purely event-driven interaction among simulated entities requires each read/write access among agents to be reflected by an event exchange, which may incur substantial overhead.

Synchronous Speculative Stochastic Agents (S^3A) is an optimistic algorithm tailored to the challenges of parallelizing the execution of SSA-driven agent-based simulations [6]. Taking inspiration from the classic Breathing Time Buckets algorithm [7], this synchronous algorithm proceeds in a window-based manner, allowing each agent to advance at most

978-1-6654-9799-2/22/$31.00 ©2022 IEEE

by one transition per window. Through this tight coupling among processors, the algorithm avoids the need to maintain a state history beyond a single old and new state per agent. Further, the algorithm aims to limit the scheduling overhead by allowing agents to access each other's states directly without mediation in the form of explicit events.

We previously showed that S^{3}A can substantially accelerate simulations of a large-scale epidemics model with a highly dynamic topology. However, due to S^{3}A's underlying assumptions, it cannot benefit from locality in the agent interactions. For instance, in epidemics models, agents are commonly situated in compartments, with each agent's interactions limited to its current compartment [8]. If communication is sufficiently local and different model portions are thus sufficiently decoupled, the asynchronous execution of Time Warp may better exploit the model's inherent concurrency.

This paper explores the suitability of asynchronous and synchronous speculative algorithms for accelerating agent-based simulations with rate-driven transitions in continuous time. We describe the fundamental differences in the synchronization algorithms and their consequences for executing SSA-driven agent-based simulations. A parametrizable epidemics model based on an agent-based formulation of the classical susceptible-infected-recovered model serves as a basis for extensive performance measurements and profiling. After identifying the model properties decisive to whether a synchronous or asynchronous is preferable, we provide profiling results to identify the remaining potentials for further performance gains. Finally, we outline an approach to combine the algorithms to exploit locality while maintaining efficiency with respect to tightly coupled model portions.

II. Analysis

The challenges for parallel execution of SSA-driven agent-based simulations are mainly due to the reliance on rate-based transitions in continuous time. In the following, we underline the need for continuous-time simulation and its specific consequences when considering its parallelization.

A. The Case for Continuous-Time Simulation

The continuous nature and lack of lower bounds on transition delays of the considered type of simulation models largely prohibits a parallelization using purely conservative algorithms. One might be tempted to sidestep this issue by discretizing simulated time and employing existing conservative methods for time-stepped agent-based simulation [9]–[11]. However, tiny time steps would be required to adequately reproduce the results of even a simple cellular automaton with rate-driven transitions.

When considering some physics-inspired models initially defined in terms of ordinary differential equations such as Social Force [12] or Intelligent Driver Model [13], the time steps represent numerical integration steps, allowing the error to be limited by a suitable choice in integration scheme [14]. In general, however, bounds are not easily available, and the error can become substantial if the time step size is too large [15].

Importantly, the step size defines a lower bound on the delay of any effect propagating from one agent to the next. Assuming

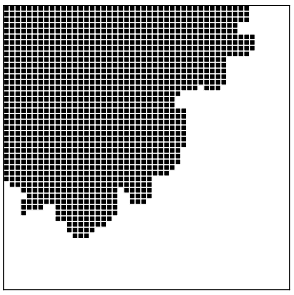


Fig. 1: Illustration of the cellular simulation on a 50×50 grid. In this intermediate state, half of the cells are still alive.

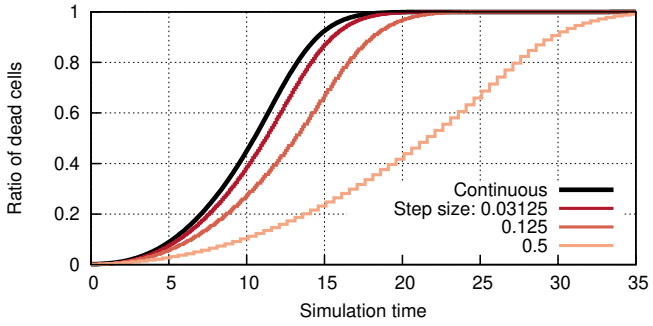


Fig. 2: Ratio of dead cells over time, showing the time-stepped execution deviation from the continuous reference.

that transitions occur at rate r, the time until the next transition is exponentially distributed with mean $1/r$. While a time-driven simulation can imitate such processes by carrying out a Bernoulli trial at each time step to determine if a transition takes place, the time step size defines a lower bound on the time between transitions. Since simulations often involve the propagation of effects across large numbers of simulated entities, very small time step sizes may be required to represent the dynamics of a continuous-time reference model adequately.

We illustrate this issue using a simple cellular model on a 50×50-cells grid. Cells carry a binary *alive/dead* state. Initially, all cells apart from the top-left one are alive. At a rate of 1, each dead cell causes all its Moore neighbors to die simultaneously. The simulation ends once all cells have died. We compare the simulation end times when executing the model in continuous time as a discrete-event simulation, and in a time-stepped fashion. Given the rate of 1, the per-step transition probability in the time-stepped variant with step size τ is computed via the cumulative distribution function of the exponential distribution as $1 - e^{-\tau}$. Figure 1 illustrates an intermediate state in which half of the cells are still alive.

Figure 2 shows the evolution of the ratio of dead cells across simulation time. The data was generated by executing 10^6 simulation runs for the continuous reference and each step size. We can see that in the time-stepped simulations, the sequence of transitions is consistently delayed compared to the continuous reference simulation, even with the smallest time step size of 0.03125. The average simulation end times and their 99% confidence intervals were 17.5 ± 0.004 for the continuous reference, and 36.7 ± 0.005, 22.0 ± 0.004,

18.6 ± 0.004 for the step sizes of 0.5, 0.125, and 0.03125.

We can see that the mean simulation end time using the time-stepped variant always exceeds that of the reference, reflecting the cumulative delay introduced by the time steps. Even at a time step size of 0.03125, the deviation was still 6%. These results demonstrate the substantial deviation that time-stepped simulation can introduce when compared to a continuous-time reference simulation. A more thorough analysis of the effects of discretization in time on continuous-time simulations is given in [16].

B. Challenges to Parallelization

The considered class of models is characterized by rate-driven transitions in continuous time across dynamic topologies, wherein each agent transition may be associated with instantaneous read and write accesses to neighbor states. These characteristics severely impede predictions of the times of future transitions, the agents at which the transitions occur, and the neighbors to be accessed. However, such predictions are the basis for lookahead, which is required to apply conservative synchronization algorithms. In the following, we argue that given the above model characteristics, efforts to extract lookahead degenerate to a speculative execution of transitions, suggesting the use of optimistic synchronization instead.

The distributions from which waiting times are drawn typically allow for transitions with zero or near-zero delay. This allows cause-and-effect chains to cascade through the simulated system within very short amounts of time. Even so, methods to compute lookahead dynamically would still allow conservative synchronization algorithms to be applied. By presampling the pseudo-random numbers used to compute transition delays, the times of future transitions could be computed ahead of time [17]. With this information, the propagation of effects throughout the model could be predicted by following the topology defined by the agents' neighborhood relationships [18], but since the transitions modify the topology dynamically, the sequence of agents to undergo transitions is not known in advance. Consequently, predictions of future transitions to extract lookahead would require a partial or full execution of transitions to determine their effects. In essence, this describes a speculative mode of execution similar to optimistic synchronization algorithms. A remaining difference is that conservative algorithms would use the precomputed effects of transitions only to determine lookahead and thus recompute some of the transitions, whereas optimistic algorithms only discard results if they have been computed in violation of the correct temporal ordering of events.

However, SSA-driven agent-based simulations pose challenges even with optimistic synchronization. Classical speculative synchronization algorithms strictly follow the discrete-event paradigm, which requires any agent interactions to be reflected by events stored in an event list and possibly exchanged among processors. This event-driven mode of execution implies considerable computational and memory overhead when considering fine-grained and instantaneous agent interactions. In the next section, we contrast a specialization of the asynchronous Time Warp algorithm to support tightly coupled agent-based models efficiently, and a synchronous algorithm tailored explicitly to such models.

III. Synchronization Algorithms

A. Asynchronous Execution using Time Warp

In Time Warp synchronization, events are executed independently of their safety. If a causal inconsistency is later detected (e.g., due to the reception of a *straggler* message received in violation of timestamp order), incorrectly-executed events are rolled back, and execution is resumed from a consistent snapshot. During rollback execution, inconsistently-generated events are annihilated by generating so-called antimessages, which can cause cascading rollbacks. This overall scheme is depicted in Figure 3.

SSA-driven simulation of tightly-coupled agents using Time Warp typically involves two main classes of events: i) *agent state transitions*, and ii) *access events* required to represent inter-agent accesses across processor boundaries in the event-driven paradigm. Access events are likely to be fine-grained to the extent that they might only entail reading a single state variable. The parallel execution of such fine-grained events may not amortize the synchronization overhead.

A second challenge is caused by the high degree of coupling among agents. Classical Time Warp simulations have been shown to be efficient [19] when the runtime environment can capture some degree of independence among the different simulated entities. In the case of tightly-coupled agent-based models, only limited independence is available since the agents may access each other's states instantaneously during a transition, which typically entails pairs of "request"/"reply" events. The tight temporal coupling among agents in the presence of large volumes of these events can cause frequent rollbacks in a Time Warp-based execution. Furthermore, each access may change the transition rates of the affected agent, which requires its next pending transitions to be rescheduled.

Two main strategies can tackle these hindrances to an efficient Time Warp-based simulation. The first strategy is related to exploiting regions of interactions among agents. If the model can be partitioned into regions so that the intra-region coupling is higher than the coupling across regions, each region can be mapped to a logical process (LP) [5].

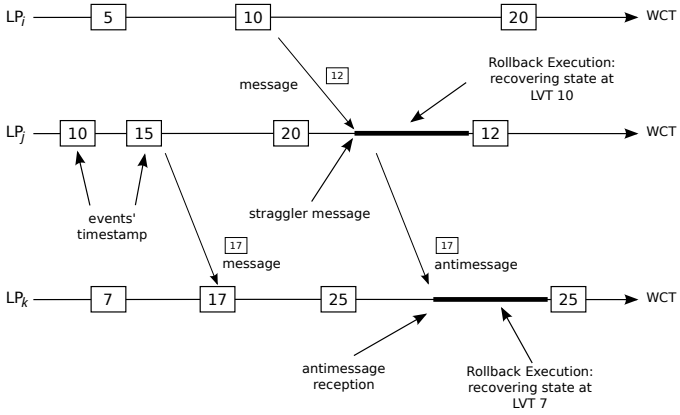


Fig. 3: Time Warp execution dynamics.

Within a region, agents can interact via direct accesses to state variables since the synchronization algorithm guarantees that all local agents observe the same simulation time upon an event's execution. Agents assigned to separate regions still interact via message passing.

As a second strategy, we specialize the traditional Time Warp runtime paradigm by introducing the concept of *retractable events*. The simulation model developer can mark an event as *tentative*. If the model takes no further action, it is managed by the runtime environment as a regular event. However, the model can also remove that event from the system or associate the same event with a different timestamp. If, after a timestamp update, the event is still in the future of the destination LP, that event can be simply moved backwards without affecting the overall simulation performance. Conversely, moving an event in the past of the current simulation time of the destination LP can be managed as if it were a regular straggler message.

B. Synchronous Execution using S^{3}A

The S^{3}A algorithm operates under the assumption that sequences of transitions may swiftly and unpredictably affect arbitrary agent states throughout the simulation. To account for this assumption, the algorithm maintains a tight coupling among processors to limit the frequency and cost of rollbacks. This is accomplished through a round-based synchronous approach using global barrier synchronizations in shared memory. S^{3}A allows all inter-agent accesses both within or across regions to be carried out by direct accesses to the agents' state variables. Since the tight temporal coupling among processors may severely limit the exploitation of a model's concurrency, each round must be as inexpensive as possible. The overhead of optimistic algorithms lies in the management of the lists of events, previous states, and antimessages and the cost of rollbacks. In S^{3}A, state saving is limited to a single old and current state per agent, similar to synchronous update schemes for time-stepped cellular automata [20], [21]. Due to the fine-grained state saving on the level of individual agents, a rollback involves only the inexpensive operation of copying the old state to the current state. Further, the algorithm avoids the need for antimessages by ruling out transitive causality violations altogether.

We describe the high-level operation of the S^{3}A algorithm based on the pseudo-code shown in Algorithm 1. For an in-depth description of the S^{3}A algorithm, see [6]. Agents are distributed to a set of processors. In each round of the algorithm, the processors execute local transitions in non-decreasing timestamp order up to a bound determined as the sum of the global minimum timestamp among all scheduled transitions and a tunable initial window size τ_0. Each transition may involve immediate read/write accesses to other agents. The key idea of the algorithm is to dynamically reduce the window size so that at the end of the round, the window contains only those transitions and accesses that can safely be committed. This is accomplished by guaranteeing that the final window contains for each agent exactly the earliest transition or access according to their timestamps, if any. By definition, these transitions or accesses can never be displaced by another transition or access.

Algorithm 1: Main loop of the S^{3}A algorithm.

```
1  while !termination_criterion do
2      global_bound ← get_global_min_ts() + τ0
3      foreach thread in parallel do
4          execute transitions earlier than global_bound, dynamically
             reducing global_bound according to Algorithm 2
5          barrier()
6          foreach agent in active_agents ∪ accessed_agents do
7              if agent.min_access_ts < global_bound then
8                  commit transition, enqueue new events
9              else
10                 roll agent back to previous state
11         barrier()
```

Algorithm 2: Agent state access in S^{3}A.

```
1  procedure Agent::access(access_ts)
2      agent.lock()
3      if access_ts < agent.min_access_ts then
4          // access is earliest in round so far
5          if agent.min_access_ts ≠ ∞ then
6              roll back agent
7              global_bound ←
8                  min(global_bound, agent.min_access_ts)
9          carry out agent state access
10         agent.min_access_timestamp ← access_ts
11     else
12         // access is deferred to a subsequent round
13         global_bound ← min(global_bound, access_ts)
14     agent.unlock()
```

Algorithm 2 shows the dynamic adaptation of the window size during an agent access as part of a transition. The variable `earliest_access_ts`, which is initially set to ∞, records the earliest timestamp of an access to the agent. If a new access arrives with a timestamp earlier than `earliest_access_ts`, the new access can be carried out, displacing any previously recorded access with larger timestamps by immediately rolling back the agent to its old state. On the hand, if an earlier access has been recorded previously, the window size is reduced to exclude the current access and its associated transition from the current round. The use of a global window rules out transitive effects of displaced or deferred accesses since any previously computed transitions and accesses with timestamps above the window bound are rolled back at the end of the round (Algorithm 1, line 10). Similarly, any newly scheduled transition within a round is deferred to a subsequent round by reducing the upper window bound to its timestamp.

IV. Benchmark Model and Implementations

We compare the asynchronous and synchronous parallel simulation algorithms using a simulation model of the epidemic spread of a contagion. The model is an extension of

the agent-based formulation [22] of the classical susceptible-infected-recovered model. We consider the SIRS variant, in which recovered agents eventually return to the susceptible state. In our model, each agent is situated in one of a configurable number of fully connected regions, each initially populated with the same number of agents. Each agent has 8 neighbors chosen uniformly at random within the same region. Hence, the number of regions determines the degree of locality in the agent interactions.

Following the SSA, transition delays are drawn from exponential distributions with static or dynamic rates. For susceptible agents, the infection rate is equal to the number of infected neighbors. Hence, agents entering or leaving the infected state must notify their neighbors so their transition to the infected state can be rescheduled according to the changed rate. The transitions to the recovered state and back to the susceptible state occur with constant rates of 1. Two additional types of transitions introduce dynamic changes to the topology defined by the agents' neighborhood relations. The first type of transition changes an agent's neighbors within its current region uniformly at random, potentially changing its infection rate or the neighbors' infection rates in the process. The second transition type migrates an agent to another region chosen uniformly at random and links the agent to new neighbors in the selected region. The rates at which these two types of transitions take place allow us to control the degree of computational load and agent interaction within each region on the one hand and the interdependence of transitions across regions on the other hand. Overall, this system resembles epidemics models as used in real-world epidemics studies [23], which aim to capture the effects of the populations' everyday and long-distance mobility.

A. Sequential Reference Implementation

Our reference sequential implementation in C++ schedules timestamped transitions using the `priority_queue` container class from the standard template library. If a transition is rescheduled before being executed, it remains in the container. Each agent's current pending transitions are stored as part of the agent state. When a transition extracted from the container is identified as outdated, it is discarded. In preliminary experiments, we found this approach to be substantially faster than explicit event retraction using the `set` container. Agent accesses are carried out by direct access to the agent states without scheduling events. Pseudo-random numbers are drawn using the Xoroshiro128** generator [24].

B. Time Warp Implementation

The model implementation in Time Warp is based on a certain number of LPs mapped to regions of the simulated space. At simulation startup, agents are evenly distributed among the LPs, forming local groups. Each agent is associated with three independent chains of events, one for each type of transition. The transition timestamps are drawn from exponential distributions using the Xoroshiro256** generator. The transitions with respect to each agent's *susceptible*, *infected*, or *recovered* state have been implemented as retractable events.

The agents' neighborhood relationships are symmetric: every time an agent randomly picks a neighbor, the other agent does the same, if space for additional neighbors is left. Once an agent receives a migration event, it disconnects itself from the hosting LPs, and an event is scheduled to a random LP in the simulation. The destination LP receives a message piggybacking the agent's current state, which, upon the creation of local links with other agents, might trigger the reevaluation of the next-transition timestamp for all involved agents. If an agent re-evaluates the next-transition timestamp, it simply informs the underlying runtime environment that the timestamp of the retractable event associated with its next state transition should be updated.

This support has been implemented within the open-source ROOT-Sim simulation framework [25]. Each worker thread implements a private queue that handles the retractable events associated with the agents bound to it. The private queue is implemented as a k-heap data structure, allowing efficient priority changes and removal of events. In order to extract a new event, each thread chooses the lowest timestamp between the normal events queue and its private retractable events queue. This strategy increases the next-event extraction cost only marginally.

C. S^3A Implementation

Our S^3A implementation is based on the same code as the sequential reference both for transition scheduling and the benchmark model. The modifications required were the introduction of multithreaded execution using pthreads, the implementation of the main parallel execution loop of Algorithm 1, and the wrapping of agent accesses to update the window bounds according to Algorithm 2. As in the sequential implementation, neighbor accesses are carried out in the form of direct memory accesses and are thus not scheduled as events. Atomic operations are used to carry out the minimum operations required to determine the initial window bounds and update the upper bound throughout each round.

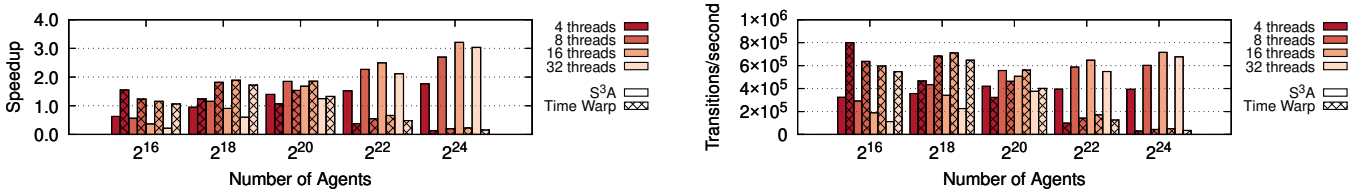
V. Experimental Assessment

A. Experiment Setup

We executed our implementations on a dual-socket machine, each socket equipped with an Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz. The total amount of RAM is 256 GB.

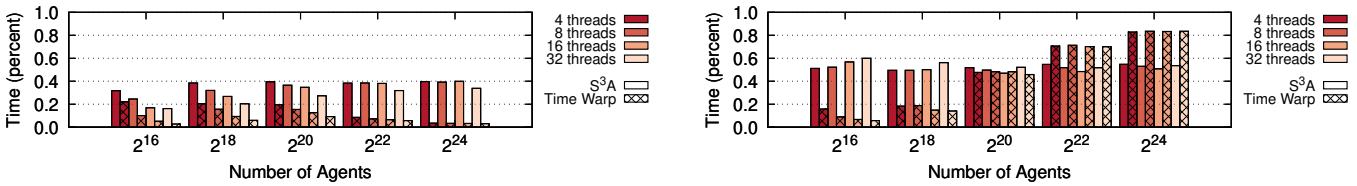
We considered two different values for the migration rate of 0.01 and 0.16 to mimic highly variable and more stable scenarios. The total number of agents was varied between 2^{16} and 2^{24}, accounting for medium and large simulation scenarios, evenly distributed across the available regions at simulation startup. The number of regions was configured as 128 and 8192, representing scenarios with low and high degrees of locality in the agent interactions. We enforced a minimum region size by discarding configurations with fewer than 128 agents.

In the following, we provide performance results comparing the performance of the S^3A and the Time Warp algorithms, as well as profiling data to highlight the causes of the observed results. Each data point represents the average of three measurements.



(a) Speedup (b) Transitions/s

Fig. 4: Performance results with 128 regions, migration rate 0.01.

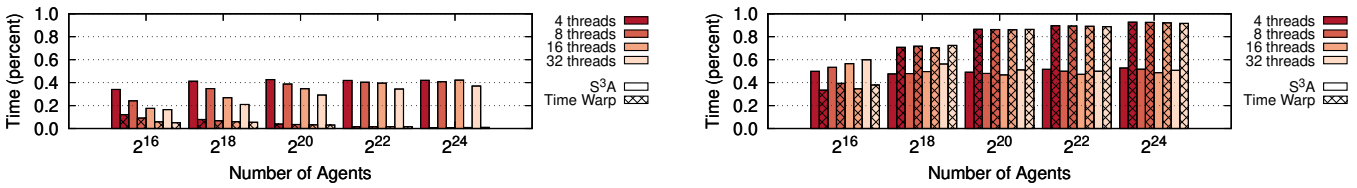


(a) Time spent in forward execution. (b) Time spent in event management.

Fig. 5: Profiling results with 128 regions, migration rate 0.01.

(a) Speedup. (b) Transitions/s.

Fig. 6: Performance results with 128 regions, migration rate 0.16.



(a) Time spent in forward execution. (b) Time Spent in event management.

Fig. 7: Profiling results with 128 regions, migration rate 0.16.

B. Performance and Profiling Metrics

In our measurements, we employ entirely separate implementations of Time Warp and S^{3}A. Therefore, it is critical to identify metrics that permit a fair comparison.

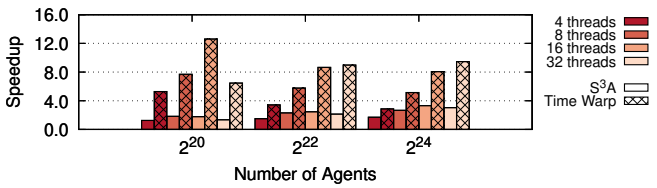
We focus on two principal metrics: the *speedup* over the sequential reference and the *throughput* expressed in terms of committed transitions per second of wall-clock time. We include only the actual simulation processing time for both metrics, i.e., we do not consider the initial model setup and final cleanup phases. Both the S^{3}A and Time Warp implementations rely on several data structures and subsystems that might require non-negligible time before starting the simulation, but optimizing these phases is outside the objectives of this paper.

The selected profiling metrics are *forward execution time* and *event management time*. The former accounts for the total time spent by both simulators running the model's code, i.e., not considering any housekeeping operations. The latter considers all the housekeeping operations related to event management, such as enqueuing new events, extracting next events, and executing rollbacks.

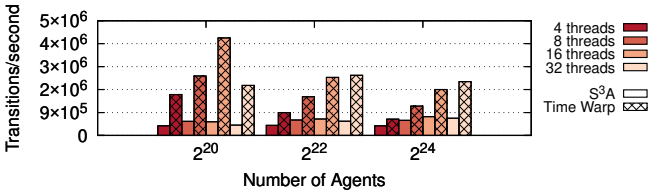
For the Time Warp implementation, we also provide classical measures, namely the *efficiency* (in terms of committed events/executed events) and the *rollback length* (i.e., the average number of events that are undone every time that a straggler message is received and a rollback operation is carried out).

C. Experiment Results

In the configurations with 128 regions, the Time Warp implementation generally delivers poor performance relative to the sequential baseline. The migration rate and the number of agents play an essential role, however: with higher migration rates (Figure 6), the slowdown is generally more apparent than with lower rates (Figure 4). Similarly, smaller agent counts deliver better speedup.

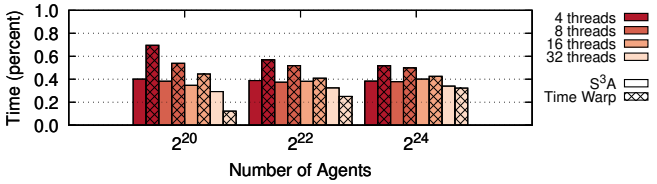


(a) Speedup.

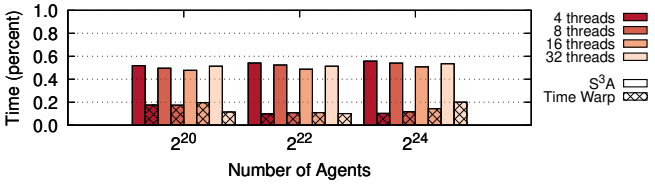


(b) Transitions/s.

Fig. 8: Performance results with 8192 regions, migration rate 0.01.

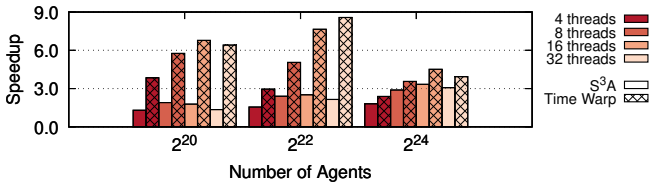


(a) Time spent in forward execution.

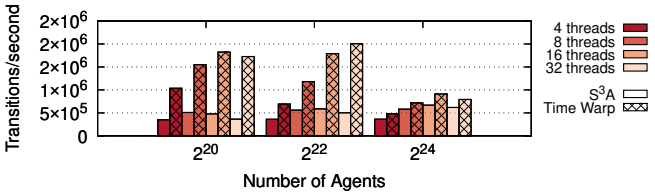


(b) Time spent in event management.

Fig. 9: Profiling results with 8192 regions, migration rate 0.01.

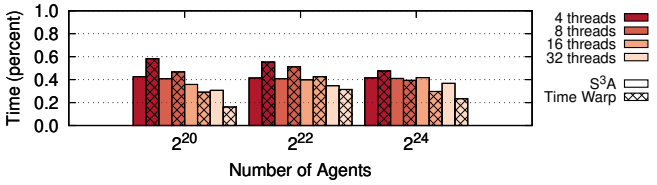


(a) Speedup.

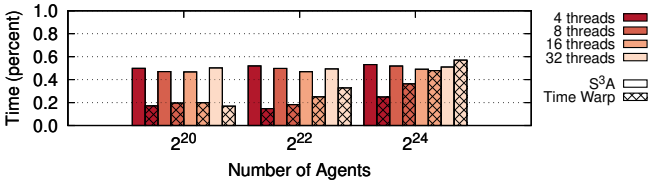


(b) Transitions/s.

Fig. 10: Performance results with 8192 regions, migration rate 0.16.



(a) Time spent in forward execution.



(b) Time spent in event management.

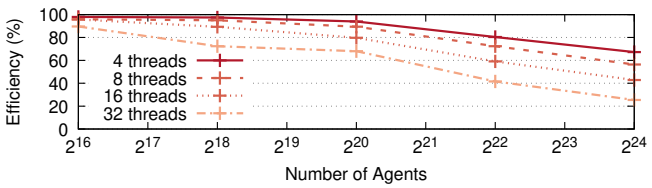
Fig. 11: Profiling results with 8192 regions, migration rate 0.16.

This behavior is mainly due to the number of agent migrations being large compared to the number of regions. Migration frequently causes a large number of expensive rollbacks: as shown in Figure 12, when the agent count increases, efficiency decreases. In configurations with 32 threads, efficiency is as low as 20%. At the same time, the rollback length can reach 350 events per rollback.

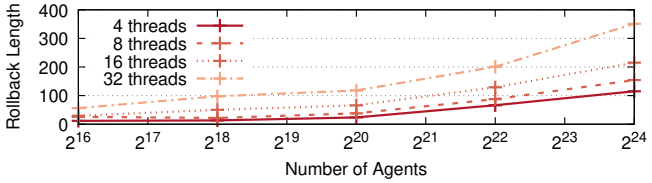
We note that the rollback length directly depends on the checkpoint frequency, which is controlled by an autonomic agent in ROOT-Sim [26]. This agent takes several internal parameters into account to tune the checkpoint interval. The relevant ones for this particular model are the event granularity, which is fine-grained on average, and the size of the LP state, which grows with the agent count. Therefore, the autonomic checkpoint strategy is selecting a large checkpointing interval in an attempt to spend more time in forward processing, trying to favor fine-grained events against costly checkpoints.

Overall, rollbacks are expensive in these configurations exhibiting few and large regions with many region interactions. As shown in Figures 5 and 7, the percentage of time spent in event management (which also accounts for rollbacks) is substantial. Rollbacks further disrupt execution by indirectly slowing down other LPs, which would be more likely to generate more straggler events. As again shown in Figures 5 and 7, the percentage of time spent running events is minimal.

Conversely, S^{3}A provides acceptable speedup. The overhead of synchronous rounds becomes apparent with low agent counts. For example, in Figure 4a, with 2^{16} agents, speedups worsen when increasing the thread count. With 32 threads, each thread only processes a maximum of about 2000 agent updates per round. Still, the results show good scaling in the number of agents, suggesting that S^{3}A performance could grow with larger agents count if a sufficient number of processor cores is available. This is supported by Figure 4, where the transition throughput increases with both the thread and agent counts. Increasing the migration rates has no impact



(a) Efficiency.



(b) Rollback Length.

Fig. 12: Time Warp performance metrics with 128 regions, migration rate 0.01.

on the speedup results (see Figure 6), whereas the transition throughput improves. The results highlight that migrations in S^{3}A are less computationally demanding than in Time Warp, as expected.

With the larger region count of 8192 the runtime behavior of S^{3}A is mostly unchanged, as visible in Figures 8 and 10. Even the profiling results, shown in Figures 9 and 11, are almost identical. This result indicates the robustness of S^{3}A's performance towards changes to the model configuration. Due to its assumptions, S^{3}A's performance is largely unaffected by the amount of locality in the agent's interactions.

In contrast, Time Warp observes a substantial improvement in performance, delivering substantial speedups and high transition throughputs. However, as shown in Figure 8, the trend seems to worsen when increasing in agent count. We conjecture that an insufficient agent density leads regions to engage in long incorrect speculative state trajectories, leading to infrequent but expensive cascading rollbacks. On the other hand, excessive agent densities approach the pathological scenario shown earlier with 128 regions.

With the migration rate set to 0.16, the best-performing results are observed at a large agent count, as shown in Figure 10. Overall, speedups are lower, but these results suggest a bell-shaped speedup behavior for Time Warp simulations. Interestingly, S^{3}A performs similarly to Time Warp in the densest configuration. The safer and controlled round-based optimistic execution of S^{3}A pays off when miss-speculations in Time Warp are either too costly (large regions state, expensive rollbacks) or too frequent (high migration rates, many straggler messages).

VI. Discussion and Research Directions

The two algorithms compared in this paper operate under contrasting assumptions. S^{3}A dynamically reduces the local window size and rolls back transitions and agent accesses so that transitive errors are avoided entirely. This strict approach is appropriate if inter-agent communication is unpredictable, global, and may occur with only small delays in simulation time. On the other hand, Time Warp allows comparatively coarse-grained LPs to advance in time asynchronously, which is appropriate if the degree of coupling among LPs is low. Following these assumptions, our measurement results have shown that Time Warp excels if there is a large degree of locality in the inter-agent communications, splitting the model into loosely coupled LPs. Conversely, S^{3}A is superior at model configurations with largely global communication.

Outside of the two extremes of global and localized communication within small regions, neither of the approaches fully aligns with the properties of the considered model. If there is only a modest number of regions, Time Warp must either divide each tightly coupled region into multiple LPs acting asynchronously or fall back to sequential execution per region. On the other hand, the synchronous execution of S^{3}A fails to exploit the locality in the inter-agent communication.

Both algorithms have room for performance improvement on their own, based on the results shown in Section V-C. The plain Time Warp algorithm could benefit on shared-memory machines by different event-management strategies, e.g., the ones presented in [27], although it should be assessed experimentally whether the limitations shown in this paper are inherent to the paradigm or dependent on the specific implementation. The scaling properties of S^{3}A could be enhanced by refinements to the thread synchronization paradigm. For example, the global minimum may be updated only periodically, trading an increase in rollbacks for a reduced rate of atomic writes.

As a further research direction, the algorithms could be combined in a layered synchronization mechanism to harness the benefits of both. Figure 13 illustrates the envisioned synchronization mechanism. Each model region is simulated using an instance of the S^{3}A algorithm, which would guarantee that at the end of each window, the local causality constraint is upheld with respect to transitions *within a region*. Each S^{3}A

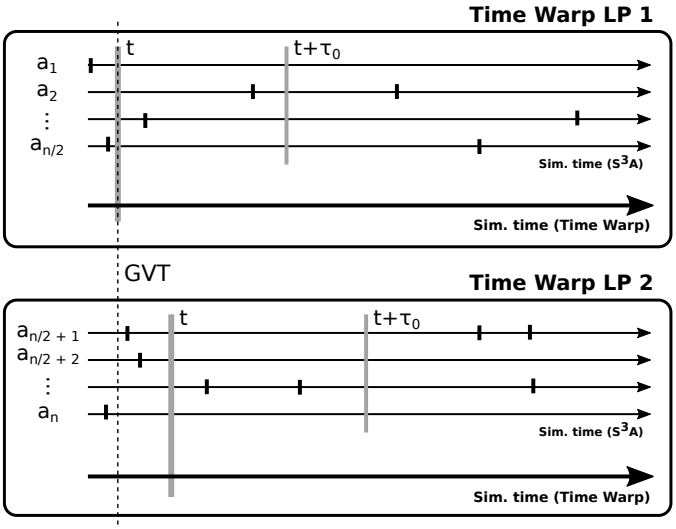


Fig. 13: Envisioned layered execution. The transitions within each tightly coupled model region are handled by S^{3}A. Each S^{3}A instance is a Time Warp LP, allowing for causality violations caused by inter-region interactions to be resolved.

instance is an LP of a Time Warp simulation. On the Time Warp level, an LP's simulation time is the lower bound of its S^{3}A instance's current window, and an event involves the advancement of the local S^{3}A instance by one time window. Since the transitions within a time window processed by an S^{3}A instance may be invalidated by a straggler event, each Time Warp LP performs state saving at the granularity of entire S^{3}A time windows. With this synchronization mechanism, S^{3}A exploits the per-region concurrency inherent to the model while accounting for the tight coupling among agents. In contrast, Time Warp exploits the large degree of decoupling across regions while still maintaining causality. Similar to the use of Time Warp on its own, this layered approach requires the identification of model portions with limited interdependence, putting a key focus on the partitioning scheme. An implementation and evaluation of this approach is part of our future work.

VII. Related Work

A variety of approaches have been explored to efficiently execute simulation models with tightly coupled simulation entities in parallel. In the following, we discuss works focusing on algorithms and on runtime environment support for such models.

A synchronous synchronization algorithm that is closely related to S^{3}A is Breathing Time Buckets (BTB) [7]. Within a synchronization cycle, LPs first compute events speculatively while withholding new events targeting other LPs. The timestamp of the earliest new event defines the event horizon up to which executed events can be committed and their associated new events can be delivered. Since this approach rules out transitive errors, it does not require the generation of antimessages as in Time Warp. In contrast to BTB, S^{3}A allows each event to involve instantaneous and direct accesses to simulation entities at other LPs, while still guaranteeing correctness of the committed events. BTB would require such accesses to represented by newly scheduled events, which would be executed over the course of multiple synchronization cycles.

In the domain of biochemical systems, a number of algorithms for parallelized execution of SSA-driven simulations have been explored. In [28] Time Warp is employed to execute the Next Reaction method, by relying on an event retraction scheme similar to the one used in our Time Warp implementation. Time Warp has also been effectively used to support the Next Subvolume Method [29]–[31] by constraining the optimism, which is a technique also employed by S^3A.

In the context of Time Warp, several works have studied techniques to allow concurrent access to the simulation state of remote LPs while executing events speculatively. Some works [32], [33] rely on the concept of state query, where LPs can access other LP's states by explicitly requesting a copy of the data via message passing. In [33], to reduce the performance penalty due to repeated queries, a single LP acts as a centralized data store, which is subject to a custom rollback procedure, while maintaining multiple versions of the variables' data. Multi-versioning of variables is also used in [34]–[36]. In [34], the model explicitly registers LPs as readers or writers on the shared variables—in our simulations, agents can act as both. Theoretical protocols to maintain replicated instances of variables are provided in [35]. In [36], substantial performance gains are achieved by avoiding some of the event exchanges required in traditional implementations.

Several approaches allow for accesses to other LPs' variables by constructing groups of LPs [37], [38]. In [37], one LP group can be active at any time since a single thread can run it in the system. Portions of the state can be marked as public and can be accessed by other groups of LPs by relying on Software Transactional Memory. The work in [38] has no notion of public attributes, but LPs can be regrouped dynamically based on runtime information regarding the frequency of mutual accesses. In [39], a mechanism based on page-fault interception is used to capture memory accesses to a remote LP at runtime, while a dedicated synchronization protocol enforces time-consistent access to the data.

Our work is differentiated from the above proposals by allowing agents to directly access each other's variables if they reside in the same region (Time Warp) or globally (S^{3}A).

VIII. Conclusions

We presented an experimental comparison of an asynchronous and a synchronous speculative simulation algorithm for tightly-coupled continuous-time agent-based models. Relying on a benchmark model with configurable amounts of locality in the agent interactions, we showed the scenarios under which each algorithm's strengths are observed. The performance of the synchronous algorithm S^{3}A was shown to be virtually independent of the amount of locality, allowing for modest speedup even with largely global interactions. In contrast, a specialization of the asynchronous Time Warp algorithm requires larger amounts of locality to achieve speedup, but vastly outperforms S^{3}A in suitable configurations.

As a future research direction, we sketched a layered combination of the two algorithms in which locality is exploited through Time Warp, whereas local tightly coupled model portions are parallelized using S^{3}A.

Acknowledgments

Financial support was provided by the Deutsche Forschungsgemeinschaft (DFG) research grant UH-66/15-1 (MoSiLLDe).

References

[1] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *The journal of physical chemistry A*, vol. 104, no. 9, pp. 1876–1889, 2000.

[2] X. R. Hoffmann and M. Boguñá, "Memory-induced complex contagion in epidemic spreading," *New Journal of Physics*, vol. 21, no. 3, p. 033034, 2019.

[3] T. Warnke, O. Reinhardt, A. Klabunde, F. Willekens, and A. M. Uhrmacher, "Modelling and simulating decision processes of linked lives: An approach based on concurrent processes and stochastic race," *Population studies*, vol. 71, no. sup1, pp. 69–83, 2017.

[4] R. M. Fujimoto, "Parallel and distributed simulation systems," in *Proceeding of the 2001 Winter Simulation Conference*, vol. 1. IEEE, 2001, pp. 147–157.

[5] D. Jefferson and H. Sowizral, "Fast concurrent simulation using the Time Warp mechanism. Part I. Local control," Rand Corporation, Santa Monica, CA, Tech. Rep., 1982.

[6] P. Andelfinger and A. Uhrmacher, "Optimistic Parallel Simulation of Tightly Coupled Agents in Continuous Time," in *IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE Computer Society, 2021, pp. 1–9.

[7] J. Steinman, "SPEEDES: Synchronous parallel environment for emulation and discrete event simulation," in *Proceedings of the SCS Western Multi-conference on Advances in Parallel and Discrete Simulation*, vol. 23, 1991, pp. 1111–1115.

[8] F. D. Sahneh, C. Scoglio, and P. Van Mieghem, "Generalized epidemic mean-field model for spreading processes over multilayer complex networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1609–1620, 2013.

[9] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, and C. Greenough, "Flame: simulating large populations of agents on parallel hardware architectures," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, 2010, pp. 1633–1636.

[10] G. Cordasco, R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, and C. Spagnuolo, "Bringing together efficiency and effectiveness in distributed simulations: the experience with D-MASON," *Simulation*, vol. 89, no. 10, pp. 1236–1253, 2013.

[11] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "A survey on agent-based simulation using hardware accelerators," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–35, 2019.

[12] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[13] A. Kesting, M. Treiber, and D. Helbing, "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4585–4605, 2010.

[14] M. Treiber and V. Kanagaraj, "Comparing numerical integration schemes for time-continuous car-following models," *Physica A: Statistical Mechanics and its Applications*, vol. 419, pp. 183–195, 2015.

[15] F. J. Willekens, "The life course: models and analysis," in *Population issues*. Springer, 1999, pp. 23–51.

[16] P. G. Fennell, S. Melnik, and J. P. Gleeson, "Limitations of discrete-time approaches to continuous-time contagion dynamics," *Physical Review E*, vol. 94, no. 5, p. 052125, 2016.

[17] M. L. Loper and R. M. Fujimoto, "Pre-sampling as an approach for exploiting temporal uncertainty," in *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation*. IEEE, 2000, pp. 157–164.

[18] R. A. Meyer and R. L. Bagrodia, "Path lookahead: a data flow view of pdes models," in *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation.* IEEE, 1999, pp. 12–19.

[19] R. M. Fujimoto, "Performance of time warp under synthetic workloads," in *Proceedings of the SCS Multiconference on Distributed Simulation*. San Diego, CA, USA: Society for Computer Simulation International, 1990, pp. 23–28.

[20] W. J. Tan, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "Causality and consistency of state update schemes in synchronous agent-based simulations," in *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2021, pp. 57–68.

[21] S. F. Railsback and V. Grimm, *Agent-based and individual-based modeling: a practical introduction*. Princeton university press, 2019.

[22] C. M. Macal, "To agent-based simulation from system dynamics," in *Proceedings of the 2010 Winter Simulation Conference*. IEEE, 2010, pp. 371–382.

[23] G. Großmann, M. Backenköhler, and V. Wolf, "Importance of interaction structure and stochasticity for epidemic spreading: A covid-19 case study," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2020, pp. 211–229.

[24] D. Blackman and S. Vigna, "Scrambled linear pseudorandom number generators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 47, no. 4, pp. 1–32, 2021.

[25] A. Pellegrini, R. Vitali, and F. Quaglia, "The ROme OpTimistic simulator: Core internals and programming model," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. Brussels, Belgium: ICST, 2012, pp. 96–98.

[26] ——, "Autonomic state management for optimistic simulation platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 1560–1569, 2015.

[27] M. Ianni, R. Marotta, D. Cingolani, A. Pellegrini, and F. Quaglia, "The ultimate Share-Everything PDES system," in *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 2018, pp. 73–84.

[28] A. P. Goldberg, D. R. Jefferson, J. A. Sekar, and J. R. Karr, "Exact parallelization of the stochastic simulation algorithm for scalable simulation of large biochemical networks," *arXiv preprint arXiv:2005.05295*, 2020.

[29] M. Jeschke, A. Park, R. Ewald, R. Fujimoto, and A. M. Uhrmacher, "Parallel and distributed spatial simulation of chemical reactions," in *22nd Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 2008, pp. 51–59.

[30] L. Dematté and T. Mazza, "On parallel stochastic simulation of diffusive systems," in *International Conference on Computational Methods in Systems Biology*. Springer, 2008, pp. 191–210.

[31] B. Wang, B. Hou, F. Xing, and Y. Yao, "Abstract next subvolume method: A logical process-based approach for spatial stochastic simulation of chemical reactions," *Computational Biology and Chemistry*, vol. 35, no. 3, pp. 193–198, 2011.

[32] A. Fabbri and L. Donatiello, "SQTW: a mechanism for state-dependent parallel simulation. description and experimental study," in *Proceedings 11th Workshop on Parallel and Distributed Simulation*. IEEE, 1997, pp. 82–89.

[33] D. Bruce, "The treatment of state in optimistic systems," *SIGSIM Simul. Dig.*, vol. 25, no. 1, pp. 40–49, 1995.

[34] K. Ghosh and R. M. Fujimoto, "Parallel discrete event simulation using space-time memory," Georgia Institute of Technology, Tech. Rep. GIT-CC-94/27, 1991.

[35] H. Mehl and S. Hammes, "How to integrate shared variables in distributed simulation," *SIGSIM Simul. Dig.*, vol. 25, no. 2, pp. 14–41, 1995.

[36] A. Pellegrini, S. Peluso, F. Quaglia, and R. Vitali, "Transparent speculative parallelization of discrete event simulation applications using global variables," *International journal of parallel programming*, vol. 44, no. 6, pp. 1200–1247, 2016.

[37] L.-L. Chen, Y.-S. Lu, Y.-P. Yao, S.-L. Peng, and L.-d. Wu, "A Well-Balanced time warp system on Multi-Core environments," in *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 2011, pp. 1–9.

[38] N. Marziale, F. Nobilia, A. Pellegrini, and F. Quaglia, "Granular time warp objects," in *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 57–68.

[39] A. Pellegrini and F. Quaglia, "Cross-state events: A new approach to parallel discrete event simulation and its speculative runtime support," *Journal of parallel and distributed computing*, vol. 132, pp. 48–68, 2019.