Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, eds.

TOWARD UNBIASED DETERMINISTIC TOTAL ORDERINGS OF PARALLEL SIMULATIONS WITH SIMULTANEOUS EVENTS

Neil McGlohon
Christopher D. Carothers

Department of Computer Science
Rensselaer Polytechnic Institute
110 8th St.
Troy, New York 12180, USA

Abstract

In the area of discrete event simulation (DES), event simultaneity occurs when any two events are scheduled to happen at the same point in simulated time. Since events in DES are the sole mechanism for state change, ensuring consistent real-time event processing order is crucial to maintaining deterministic execution. This is synonymous with finding a consistent total ordering of events. In this work, we extend the concept of virtual time to utilize an arbitrary-length series of tie-breaking values to preserve determinism in parallel, optimistically executed simulations without imposing additional bias influencing the ordering of otherwise incomparable events. Furthermore, by changing the core pseudo-random number generator seed at initialization, different orderings of simultaneous events can be observed, allowing for deeper statistical analysis. We implement and evaluate this mechanism using the Rensselaer Optimistic Simulation System (ROSS) and discuss the importance of deterministic event ordering given the existence of event ties.

1 INTRODUCTION

Discrete event simulation (DES) is an effective method for simulating a wide variety of phenomena. It establishes any occurrence in the simulation as being an event that occurs on a simulation entity. This event alters state and occurs at a temporal coordinate in the virtual simulation time. Because simulation state only changes when an event happens, the outcome of a simulation is entirely dependent on the initial state and the set of ordered events that alters it.

It is expected that two identically configured simulations will produce identical results. Simulations with this property are referred to as deterministic and is not uncommon for models developed in DES. The design of a general DES engine, thus, may be of broader utilization if it can provide deterministic execution. Maintaining determinism in a simulation can help confirm that there is not additional errant, undefined behavior occurring. The defined rules of a deterministic simulation, given a singular input, result in a singular answer.

From a high-level view, maintaining determinism appears to be a simple task: process events in the order in which they are scheduled to occur. However, there are complications that can add complexity to this objective. In a parallel processing environment, a simulation is partitioned across multiple processors, with events occurring between entities that may not exist on the same processor. Because of this, the nature of inter-processor clock drift and individual processing delays, ensuring a deterministic ordering of event execution requires additional synchronization overhead.

Fortunately, this is generally a solved problem in Parallel Discrete Event Simulation (PDES) with various synchronization methods for conservative parallel and optimistic parallel execution. One complication, however, is event simultaneity: events that occur at the same coordinate in virtual time. Given two simultaneous events that each result in different, non-commutative, state changes on the occurring entity,

978-1-6654-3311-2/21/$31.00 ©2021 IEEE

final simulation results will naturally depend on the ordering of these two events. To make matters worse, the emergent existence of simultaneous events may not be obvious to a model developer – nor the potential consequences such as loss of determinism.

In a parallel environment, an inter-processor event may arrive at the receiving processor after an earlier timestamped event had been processed, violating the virtual time-guaranteed happens-before relation of the two events. Correction methods like reverse-computation or checkpoints can be used to revert the simulation to the last known safe state and ensure that a consistent and valid causal ordering is maintained despite this phenomenon. However, there could be a problem when two events are to occur at the same coordinates in virtual space-time. Given the encoded standard virtual timestamps alone, there is not a clear way to define which event happens-before the other, which results in a loss of determinism.

To regain assured determinism, a set of rules can be defined dictating the expected ordering of two simultaneous events. Given a strictly increasing event counter on each processor that increments with every new event, then in the case of simultaneous events, we can give priority based on the sending processor ID. If that, too, is identical, then we can give priority based on the event ID encoded from the event counter. Determinism was regained but with one major caveat: the resulting ordering is influenced by the bias of the chosen rule-set.

Because of this bias imparted by the ruleset, it is difficult to know for certain if the resulting simulation behavior is exemplary of the simulated model's behavior as a whole or a special case determined by said ruleset's influence on event ordering. Additionally, without a deterministic total ordering of events in a parallel simulation, it is much more difficult to show that the parallel simulation is semantically the same as if it were sequentially executed. In this work, we extend the definition of virtual time to include a set of tie-breaking values and devise a mechanism for unbiased arbitration of simultaneous events in a parallel discrete event simulator with optimistic execution. This is accomplished by utilizing a rollback-safe tie-breaking uniform random number generator. We develop this mechanism into the Rensselaer Optimistic Simulation System (ROSS) and exhibit its effects on simulation performance and model behavior. Because ROSS, even in optimistic execution, utilizes fully deterministic pseudo-random number generators (PRNGs), different RNG initialization seed values will yield different but deterministic event orderings, even in the presence of simultaneous events, allowing for deeper statistical analysis of simulated models.

We propose and argue three new mechanisms for finding a deterministic total ordering of events in a parallel discrete event simulation with simultaneous events:

1. Unbiased random total ordering of otherwise incomparable simultaneous events.
2. Biased random total ordering of otherwise incomparable simultaneous events, including events created with zero-delay/offset.
3. Unbiased random total ordering of otherwise incomparable simultaneous events, including events created with zero-delay/offset,.

2 PROBLEM DEFINITION

For this work, we specifically utilize the Rensselaer Optimistic Simulation System (ROSS) PDES engine (Carothers, Bauer, and Pearce 2000; Carothers, Bauer, and Pearce 2002) which implements the Time Warp protocol (Jefferson and Sowizral 1985; Jefferson, Beckman, Wieland, Blume, Loreto, Hontalas, Laroche, Sturdevant, Tupman, Warren, Wedel, Younger, and Bellenot 1987; Jefferson, Beckman, Wieland, Blume, and Loreto 1987) in conjunction with virtual time (Jefferson 1985). In a ROSS simulation, entities or agents are represented as Logical Processes (LPs). These LPs are mapped to the various Processing Elements (PEs) that may exist. Typically, there is one PE per physical MPI process that is participating in the actual execution of the simulation.

ROSS is capable of being run in three main modes of synchronization. When the simulation is to be executed on a single process, it is executed in *sequential* mode. Maintaining determinism in this mode is trivial. The challenge of determinism comes when additional PEs are added to the simulation. With additional PEs comes the advent of *remote* events, those whose destination is on a different PE than its source.

All propositions in this work are based on the following assumptions and definitions:

Assumption 1. *In a parallel environment, arrival order of inter-processor messages is not guaranteed to be consistent across executions of a simulation.*

This is due to natural differences in local clock-speed. Even the smallest perturbation can lead to enough clock drift to change the order in which any two events from different processors may arrive at a third processor. If this is not accounted for, it can lead to drastically different final results. Fortunately, this is addressed by Time Warp's virtual time approach to parallel event processing.

Assumption 2. *Every event in the simulation is encoded with a standard virtual timestamp specifying when in simulation time the event occurs.*

Without virtual time, a receiving process will have no ability to determine when in the future the event should occur. Given this encoded virtual time, it is possible for the receiving process to deduce a partial ordering of this received event and others that the process is aware of.

Assumption 3. *Each processing element in a PDES system has a priority queue maintaining a proper order of arrived events to be processed.*

Because different LPs across PEs may be scheduling events at various times in the future, we cannot assume that events will arrive in the exact order that they are to be executed – and by Assumption 1, can not be guaranteed even if one tried to make it so. Thus, each PE should manage a queue, maintaining a proper order in which events should be processed. This is almost certain to exist in any parallel simulator that implements virtual time.

In an optimistically executed simulation, events that arrive at an LP after its current definition of 'now' are referred to as *straggler* events (Fujimoto 1990). To address this, optimistic simulations can be rolled back. After rolling back the simulation to a known safe state, the undone events can be correctly ordered in the priority queue and re-processed. None of this can be accomplished without some ordered queue.

Assumption 4. *No LP can create events at a virtual time in the past relative to its own definition of virtual time.*

This is a safe assumption as the entirety of discrete event simulation relies on the basis of maintaining causal-order; causal ordering can be formally stated as:

Definition 1 (from (Jefferson 1985))**.** *Event A causes B ($A \rightarrow B$) if there exists any sequence of events $A = E_0, E_1, \ldots, E_n = B$ such that for each pair E_i and E_{i+1} of adjacent events either (a) E_i and E_{i+1} are both in the same process and the virtual time of $E_i < E_{i+1}$ or (b) event E_i sends a message to be received at event E_{i+1}.*

Because, in virtual time simulations, the virtual space-time coordinates of an event define the happens-before and causal relations of events, the only instance where two events are considered *incomparable* is when they have identical virtual timestamps, i.e. simultaneous events or an *event tie*.

Definition 2. *Two events are considered comparable if a consistent happens-before relation can be inferred between the two. If no happens-before relation can be inferred, then they are considered incomparable.*

Definition 3. *An event created with zero virtual-time delay between its encoded timestamp and the creating LP's definition of 'now' (timestamp of the causal event) is referred to as a zero-offset event.*

Assumption 5. *Every LP has at least one, rollback-safe, pseudo-random number generator stream solely dedicated to generating uniform random values to encode into events that it creates.*

To achieve the goal of creating an unbiased ordering of simultaneous events via a pseudo-random number generator, it is imperative that this stream be completely independent of other streams accessed in the model and deterministically rolled back. For reasons which will be explained in detail in Section 3.2, it is also critical that this PRNG can be deterministically rolled back to a previous state should the events it generated values for be canceled or rolled back.

With the above assumptions and definitions, we posit the following problem:

Problem Statement 1. *Given an optimistically or conservatively executed parallel discrete event simulation with the above assumptions, consisting of a partially ordered set of virtual-time encoded events E which may or may not contain event ties, find a mechanism which ensures unbiased total ordering of E such that any subsequent executions of the simulation are deterministic.*

3 EVENT SIMULTANEITY

Another way to phrase Problem 1 is: given a simulation of events, find a mechanism to randomly establish the partial ordering found by a parallel simulation in a way that is deterministic and identical to the total ordered version found by a sequential execution of the same simulation. If all events in a partially ordered simulation are comparable to each other, then the partially ordered events are also considered totally ordered.

As briefly mentioned in Section 1, deterministic ordering of simultaneous events can be achieved by establishing a ruleset that is enforced by each PE dictating which event happens-before another. If this ruleset is precise enough to render every possible event in the simulation comparable to every other event, then a deterministic total ordering can be established from its partial counterpart. In Figure 1, events A, B, C and D all occur simultaneously in the simulation. How this ruleset is defined will specify how these events are ordered to occur in the simulation. The orderings of C and D are specifically of consequence due to them each operating on the same LP.

3.1 Ordering Bias

This ruleset would, in effect, turn incomparable, simultaneous, events into comparable events that occur *simultaneously yet infinitesimally before or after one another*. An example ruleset would be as follows, in decreasing order of importance, cascading down to break any subsequent value ties.

Ruleset 1 Ordering with Explicit Bias

Give priority to:

1. Events with lower virtual timestamp
2. Events with lower PE ID
3. Events with lower LP ID
4. Events with lower (per PE) event ID

Ruleset 2 Ordering without Bias given 2-way event ties

Give priority to:

1. Events with lower virtual timestamp
2. Events that win a coin flip

Ruleset 3 Ordering without Bias given n-way event ties

Give priority to:

1. Events with lower virtual timestamp
2. Events with lower tie-breaking value

This, however, will result in a single final result with likely few options that would allow users to explore alternate orderings. Perhaps with greater consequence is the amount of bias that this ruleset instills into the simulation. In the case of a timestamp tie, it will always give priority to whatever event came from a lower numbered processor. This means that LPs mapped to lower numbered processors, for instance, will always take precedence should they tie and that the same simulation but with different numbers of PEs or a different mapping will likely generate different final results.

Given the possibility of a two-way event tie, it is possible to relinquish the simulated model from the influences of our bias by instead relying on a fair coin flip.

With this ruleset, given two tied events, A and B, there is a 50% chance of being ordered as [A,B] and a 50% chance of being ordered as [B,A]. If we can randomly generate the values necessary for this coin flip to occur, then we can, without bias, order all events in a simulation with at most 2-way event ties.

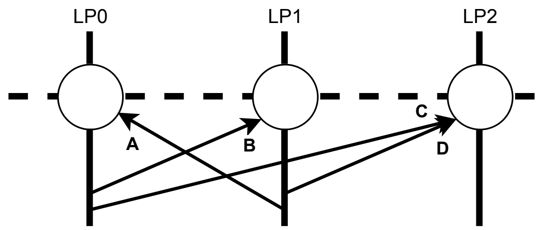


Figure 1: In this LP-Event diagram, all events: A, B, C, and D occur at each of their respective LPs simultaneously. Determining how these events, C and D in particular, should be ordered in a final total ordering may have an effect on final simulation results.

This can easily be extended to n-way ties. Instead of a coin-flip, encode an i.i.d. uniform random (UR) value into an event when it is created and implement the following ruleset.

Given an n-way tie, each event having its own independently generated uniform random value, any specific ordering of these events constitutes a single possibility (occurring with probability $1/n!$) out of all possible permutations of these tied events. Furthermore, the probability of any two tied events being ordered a specific way with respect to each other is 50%. This method is equivalent to what is described and proven in (Cormen, Leiserson, and Rivest 2001) as `PERMUTE-BY-SORTING` yielding the following lemma:

Lemma 1 (From (Cormen, Leiserson, and Rivest 2001))**.** *Given a list of length N with N distinct i.i.d. uniform random values, a uniform random permutation can be found by assigning one value to each item in the list and sorting according to these values.*

If the probability of tie-breaker collision is negligible, then it is possible to generate a uniform, or unbiased, random permutation by sorting based on the uniform randomly generated key. It follows that by using Ruleset 3, we can extract an unbiased random ordering of an n-way event tie.

One may question whether any random ordering of these events is safe. The most important thing in ordering simultaneous events is to uphold event causality: no event caused by another can happen before the other event.

Lemma 2. *If there are no zero-offset events in a simulation, then no events participating in a tie caused any other events in said tie.*

Proof. Given the contrapositive: if events participating in the tie did cause other events in the tie, then there are zero-offset events. Suppose an event participating in the tie caused another event in the tie. Because both events are participating in the tie, the virtual time difference between the two would be zero. By definition, the caused event was scheduled with zero-offset. Thus, by contrapositive proof, if zero-offset events are not allowed, then no events in an event tie caused any other events in the same tie. □

A consequence of this lemma is that if there are no zero-offset events in a simulation, then because there is no causal linking between any events participating in the tie, no two events in the tie are ordinarily comparable. If they were comparable by their virtual timestamps, then they would not be in this tie in the first place.

Because events that are not causally related to each other and incomparable could be processed in any order without violating causality, then a random permutation of these events constitutes a valid ordering.

Proposition 1. *Providing, in addition to the standard virtual timestamp, a secondary uniform random value and comparing tied events based on that will yield an unbiased ordering of events without violating causality given no zero-offset events.*

Proof. Lemma 2 assures that no events in any event tie will have a causal relationship with any others in the same tie. Because these events, given their standard virtual timestamp, are thus incomparable, any possible permutation of these events is causally safe, and by Lemma 1, an implementation of Ruleset 3 will allow these events to be comparable and yield an unbiased ordering of events. □

For clarification, we define the combination of a virtual timestamp and any secondary values determining event priority as a *virtual time signature*. Events created in accordance to Proposition 1 that are incomparable by their virtual timestamps are comparable by their virtual time signature.

It is important to note, however, that Lemma 1 relies on *distinct* values from its uniform random number generator. By the nature of PRNGs, this is not possible. For the purposes of our work, though, we utilize sufficiently precise PRNGs with very large periods, and thus we assume the probability of generating non-distinct values is negligible.

Should this probability still be considered too high, additional tie-breaking values can be generated and encoded via supplementary, independent, PRNG streams and applying `PERMUTE-BY-SORTING` recursively to sub-lists containing PRNG value collisions. This effectively increases the bit precision of the primary tie-breaking value, making overall collision even less likely while maintaining the expected lack of bias.

3.2 Determinism with Randomness

We have now established that given a good choice of ruleset, we can create *an* unbiased ordering of events in a simulation. But this alone does not grant determinism. The ROSS simulator guarantees Assumption 5, allocating an independent PRNG stream on every LP with the sole duty of generating the tie-breaker values encoded into each event that that LP creates.

One valuable property of PRNGs is that given a specific seed, the sequence of values generated from it is deterministic. In a sequentially or conservative parallel executed simulation, this allows for pseudo-randomness without sacrificing determinism. In neither of these execution modes is there ever a rollback to a previous simulation state.

When executing a parallel simulation optimistically, however, the simulation must be tolerant of rollbacks. If the state of a PRNG stream, indicating the position of the 'next' value in the sequence, is not also rolled back when necessary, then a different set of numbers will be generated for events when the simulation is resumed. As a result, LP state changes and decisions based on that PRNG stream will differ from an execution where said rollbacks did not occur – a loss of determinism.

Corollary 1. *Using a rollback-safe tie-breaking uniform random value in execution of Proposition 1 will yield a deterministic unbiased ordering of events without violating causality given no zero-offset events.*

4 ZERO-OFFSET EVENT SIMULTANEITY

Proposition 1 makes certain assumptions about the nature of the problem and the simulation environment it operates in. Specifically, it does not allow for zero-offset events to exist in the simulation.

To allow for zero-offset events in a simulation, there are some finer details that must be established to ensure that a deterministic, valid, unbiased total ordering of events can be found. Part of what makes this difficult is that the act of creating zero-offset events, by definition, creates event ties. As a result, an event that generates a new event with a zero-offset is creating a new event that, in virtual time, occurs at the same time as the event that created it. In this instance, the order in which these two events are committed to simulation history is very important.

It's also important to note that while Assumption 4 is standard for virtual timestamps, the same must also be held true for any happens-before relations inferred from an extended definition of virtual time. Furthermore, the virtual time signature of any event must be strictly greater than the time signature of its causal parent.

Event	(time, tie-breaker)
A	$(1, 0.1)$
B	$(1, 0.15)$
A'	$(1, 0.40)$
B'	$(1, 0.30)$
A''	$(1, 0.20)$

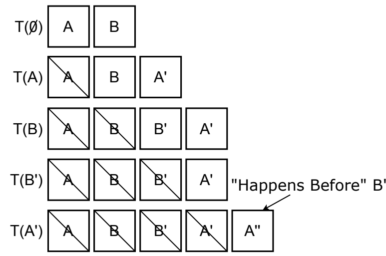


Figure 2: Diagram showing event processing queue state while processing the simultaneous events given in the table on the left. $T(-)$ represents the time in the simulation after a given event has been processed. Event A'' is created by A' but, according to its tie-breaker value, should happen before B', which has already been processed as designated by a slash.

4.1 Challenge of Zero-Offset Events

We have shown with Proposition 1 that an unbiased total ordering of simultaneous events can be found with a uniform random value acting as a tie-breaker. This works by establishing an unbiased hard rule of how any two events should be ordered in the simulation. Proposition 1 worked because any arbitrary ordering of otherwise incomparable events will result in a valid simulation.

This becomes tricky once zero-offset events are introduced. Because a parent event and its zero-offset child have the same virtual timestamps but one event *definitely* caused the other, the ordering in which these two events should be ordered in the final simulation history cannot be arbitrary.

In a solution using Proposition 1, to create an unbiased random ordering of events, an absolute comparison of i.i.d. random tie-breaking values is utilized. Given an event E that generates a child event E' with zero-offset, it is entirely possible that the uniform random tie-breaking value of E' could be less than that of event E. By our extended definition of virtual time, this would imply that event E' *happened before* event E which is impossible since event E directly caused E'. This contradiction caused by a zero-offset event is exactly the same as what would happen if a time-traveling negative-offset event were created. This establishes the following:

Lemma 3. *No event can be created in a way that would classify it as happening before any events that caused or happened-before it.*

If all we consider are their standard virtual timestamps, the two events are incomparable. However, they *are* comparable by the ordered chain of causality. Definition 1 does not account for zero-offset events; if we are to have a stable simulation with zero-offset events, we need to extend our definition of virtual time to ensure that the virtual time signature of $E < E'$ and so on for any descendants of event E'.

Definition 4. *An event that shares a standard virtual timestamp with another but has a lower tie-breaking value is defined to happen infinitesimally before the other. The converse implies that the other event happens infinitesimally after the first.*

Definition 5. *A zero-offset event, sharing the virtual timestamp with that of its parent, is considered to happen infinitesimally after its parent. The converse implies that the parent happens infinitesimally before its child(ren).*

To demonstrate the challenge of utilizing the tie-breaking mechanism we have established in Proposition 1, let us assume that we have some ability to ensure that Lemma 3 is maintained while using the same tie-breaking mechanism as before. Then we have nothing to prevent the possibility of generating events with virtual time signatures to *happen infinitesimally before* others that have already been processed.

Consider a sequentially executed simulation; there is no ability to roll-back events should a new one be generated that *happens before* one previously processed – if a mechanism cannot be used for generating a valid ordering in sequential, then it is unlikely to consistently work for parallel executions either. As an example, two events, A and B, each generate a zero-offset child event: A' and B'. A' generates a single zero-offset child, A''. The events are each scheduled to occur at the same point in virtual time but have

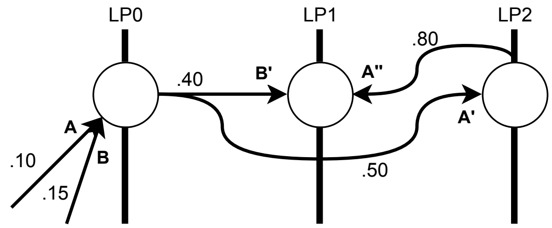


Figure 3: LP-Event Diagram depicting events all simultaneously occurring with the same virtual timestamp but depicted with varying legal tie-breaking values encoded using the additive scheme described in Section 4.2. The final ordering in the simulation for these tied events is: $[A, B, B', A', A'']$

uniform random generated tie-breaking values to determine the order of events not directly caused by each other; the virtual time and tie-breaker values for these events are shown to the left in Figure 2.

In this sequential simulation, we would start by processing events A and then B. This would schedule two new zero-offset events A', and B'. We could process these two in order according to their tie-breaker values without issue (B' first). However, after processing A', another child event is created, with a tie-breaker value that is lower than B'. The simulation processing queue state for this example can be observed in Figure 2. The result of this is an irreparable error in the simulation. In a sequential execution, an effective negative-time event was received, breaking Lemma 3. In an optimistic execution, we will enter an infinite rollback scenario and cause instability in sequential.

This is exactly the challenge of creating a fair deterministic ordering of zero-offset events: it is not possible for the simulator to know if a zero-offset event will create another zero-offset event until after its processed. Thus, how can a fair random ordering of zero-offset tie events be determined?

4.2 Biased Random Causal Ordering

It is now clear that a single pair of randomly defined tie-breaking values is insufficient for ordering two events given the existence of zero-offset events. It would always be possible to violate causality and that should *never* be possible; any scheme to determine ordering two events must guarantee causality is maintained.

We know, however, that a simulator that utilizes standard virtual time can generate a deterministic total ordering of events if no ties exist because all events have a unique time, and the final ordering should just be monotonically increasing by their timestamp. This is possible because when new events are created, they are encoded with an offset: an additive 'time from now' value. So long as that value is not negative, it guarantees causality is maintained; as long as the value is positive, it guarantees determinism.

What if, when generating a zero-offset event, instead of using a random tie-breaking value from (0,1), we generate that random value but *add* it to the random tie-breaking value of the parent event. The new tie-breaking value will be strictly greater than the parent, assuring that any comparison of the two will always order the parent first. A LP-Event diagram using the same generated values in the previous example but adding consecutively generated tie-breaking values when creating child events is shown in Figure 3.

Proposition 2. *Providing, in addition to the standard virtual timestamp, a secondary value defined by summing deterministic uniform random values generated by causally related zero-offset events and comparing tied events based on that will yield a randomized-but-biased deterministic ordering of events, including zero-offset events, without violating causality.*

Proof. Given four simultaneous events A, B, C, and D where $A \rightarrow B$ and $A \rightarrow C \rightarrow D$ and B, C, D are zero-offset from A with additive tie-breaker values. Assume that an event D is an event that violates Lemma 3 in conflict with a previously processed event B. The tie-breaker value of D is strictly greater than that of its parent: C. The only way for D to be created so that it happens before B is if C also happened before B. If C happened before B, then no ordering of B and D would violate Lemma 3. Thus, by contradiction, additive tie-breaking values will safely allow for zero-offset event tie-breaking without violating causality established from Lemma 3. □

This does allow for randomness to be introduced in the ordering of events but it introduces some potentially unintended bias into the comparison of tie-breaking values. For example, given two events scheduled for the same time: A and B. Event A creates a zero offset child A' who creates a zero offset child A''. The four events are each scheduled for the same virtual time. If the tie-breaker value for descendants is additively generated, then the probability that event A'' comes before event B is not 50%. it is possible that, based on the tie-breaking value, that A'' happens before B, but it is far more likely that B happens before A'' because the expectation for its tie-breaker value is lower than that of A''.

This is because the tie-breaking value encoded into events A' and A'' are no longer a uniform random value from (0,1) but is instead sampled from an Irwin-Hall distribution (Johnson, Kotz, and Balakrishnan 1995). The expected value of an Irwin-Hall value is proportional to the number of uniform random values added to generate it, and thus the more zero-offset descendants recursively created, the less likely they are to happen before another, independent, event. In a way, this may seem natural, but it is important to remember that these events are all scheduled by the model to happen at the same time.

This method will assuredly result in a deterministic ordering and this ordering will be somewhat randomized based on the simulation's input seed. Is there, however, a fairer way to break ties within zero-offset events?

4.3 Unbiased Random Causal Ordering

When trying to find a fair way to order simultaneous events, something to consider is whether it even makes sense for other events to interject between an event and its zero-offset child. The two related events are supposed to occur simultaneously with the exception of one being causally dependent by the other (the child *happens infinitesimally afterward*). Our next proposition is, thus, based on the following assumption:
Assumption 6. *The only event that can interject between a parent event and its zero-offset child is another zero-offset child from said parent event.*

What made the additive tie-breaker value in Section 4.2 successful in establishing a deterministic total ordering was that it guaranteed that a zero-offset child event will have a value strictly greater than its parent. It also established a unique timestamp that will not – should there be more descendants – violate some established order of other already processed events.

We can devise another mechanism that will ensure that any subsequent zero-offset children of a parent event be processed before other independent events (with a greater tie-breaking value than said parent). Consider ordering two sequences of numbers. There are numerous ways to pick some relation and choose which should be considered first in an ordering of the two sets. One common way is through *lexicographical ordering*. Just as one would find where to place new words into an English dictionary, we start by comparing the first items in the two sets. Should those two match, we recursively compare each of the subsequent items in the sequences until we find a pair that is different. Determining which of the two is 'less' than the other, lexicographically, is then based on that final comparison.

For example, let us consider two sequences $S_1 = [5,3,9,3]$ and $S_2 = [5,3,4,6]$. Lexicographically, $S_2 < S_1$ because in the third component-wise comparison, $4 < 9$. We also observe that it does not matter how long the sequence of S_2 is; as long as those first three numbers remain in each sequence, then $S_2 < S_1$ *always*.

We can extend our definition of virtual time again to include – not a single tie-breaking value but – a *sequence* of tie-breaking values. The sequence is comprised of the tie-breaking values of historical zero-offset parental events. Whenever a new event is created, a new tie-breaking value is generated and appended to the back of this running list. When a regular-offset event is created, the sequence is discarded and then the newly generated tie-breaking value for the current event is added as the sole value.

When comparing two tie-breaking sequences of different lengths but all N components of the shorter one match the first N components of the longer sequence, priority is given to the shorter one as this can only happen if the event owning the shorter sequence *caused* the other. Referring back to the English dictionary analogy, the single-letter word 'A' comes before 'aardvark' but 'aardvark' comes before 'apple'.

The result of this extension allows for us to make every single event in the simulation comparable to one another via the new virtual time signature. In this case, the virtual time signature is defined as the sequence of the virtual timestamp followed by the regular-offset tie-breaker and all consecutively generated zero-offset tie-breakers. Because of the benefits of lexicographical ordering, causality is guaranteed as any zero-offset descendant's time signature will be strictly greater than its parent, grandparent, etc. The happens-before comparison of any two tied events that are unrelated will, as in Proposition 1, be based on a comparison of two uniform-random values.

Additionally, the comparison of any two tied events that *are* related will also be based on a comparison of two-uniform random values *unless* they are causally locked to a single relative ordering because one is a descendant of the other, yielding:

Proposition 3. *Providing, in addition to the standard virtual timestamp, a secondary sequence of values defined by deterministic uniform random values generated by causally related zero-offset events and comparing tied events lexicographically based on that will yield an unbiased random deterministic ordering of events, including zero-offset events, without violating causality.*

Proof. The exact same logic as demonstrated in the proof of Proposition 2 can be applied here to guarantee determinism as lexicographical ordering provides the same strict less-than/greater-than relation between two events. Using the same events from that example, however, we can observe that should C happen before B, then D assuredly happens before B as it *happens infinitesimally after* C. The determination of the ordering of D and B still falls down to a fair comparison of two uniform random values: the two generated for B and C. The probability of D happening before B is, effectively, randomly determined by a fair coin flip and is unbiased. □

We recognize that the restriction imposed by Assumption 6 detracts from the generality of Proposition 3 and relaxing this assumption is a point of continuing research.

5 MODELS OF EVALUATION

We have evaluated our proposed solution with various methods to observe its capability as well as the performance impact of the solution's overhead. This was accomplished through strong scaling and qualitative studies with different simulated models run on ROSS with and without the unbiased tie-breaking mechanism described in Section 4.3. Additionally, we have run a number of simulations using the CODES high-performance computing interconnect network simulation framework which uses ROSS as its simulation engine to showcase the types of simulations that the tie-breaking mechanism enables (Cope, Liu, Lang, Carns, Carothers, and Ross 2011).

5.1 Scaling and Qualitative Studies

In the simple case, we observe the performance impact on a `PHOLD` simulation benchmark model. `PHOLD`, does not produce significant numbers of event-ties so the benefit of the tie-breaker is not observed and allows us to estimate the expected impact of the tie-breaking overhead (generating and encoding sequences of random values). Results of this preliminary model experiment are available in a technical report available on ArXiV (McGlohon and Carothers 2021).

But the original intention of the tie-breaking mechanism was not to improve the performance of models without event ties; it was to guarantee determinism in parallel simulations despite the existence of event-ties. To test this objective, a couple of new benchmark simulation models were developed: `Event-Ties` and `Event-Ties-Stress`. The first generates chains of events with random destinations and integer timesteps. Furthermore, subsequent events with zero-offsets are created by each received event to a degree before new incremented integer delay events are created. This produces a very high degree of event-ties and is enough to elicit non-determinism in a standard ROSS simulation without the tie-breaking mechanism. With the use of the tie-breaking mechanism, determinism was maintained.

The `Event-Ties-Stress` model follows a similar procedure to the `Event-Ties` model but instead of chains of zero-offset events it produces *trees* of zero-offset events.

These studies provide an interesting perspective into how simultaneous events, especially those with zero-offset, can be a challenge to simulate. These models exhibit the benefits that a deterministic tie-breaking solution can provide. Detailed model behavior is discussed at greater length in the associated technical report (McGlohon and Carothers 2021).

5.2 Network Simulation Modeling

One area of study that lends itself well to PDES is high performance computing interconnect network simulation. All interactions between entities in the network, such as packet routing, can be modeled as events in ROSS. The CODES network simulation framework allows for the study of massive scale simulated networks facilitating billions of simulated packets. Different technologies, such as adaptive routing, Quality-of-Service techniques, and hypothetical network topologies can be explored with this framework.

To orchestrate these simulations, CODES models use many different types of PDES events which, due to their high frequency, are prone to simultaneity. To make matters more difficult, many types of events may be configured with zero-offset. To overcome the problems, like non-determinism, that come with simultaneous events, CODES attempts to *avoid* event-ties by incorporating small additive noise to all created events. This, however, is a stopgap measure and cannot *guarantee* that event ties cannot occur. There are more caveats as well; similar to the issues discussed with the mechanism described in Section 4.2, this additive tie-avoidance scheme will impose bias into how the simulation will order events that otherwise would have occurred simultaneously.

Lastly, and potentially most importantly: by adding additive (non-zero mean) noise to the timestamps of all events, packet metrics and all measurements based off them and other event timings will be influenced by this incremental noise.

An unbiased tie-breaking mechanism allows us to explore different possibilities of orderings of simultaneous events, sampling from the space of all possible simulation occurrences for the given model.

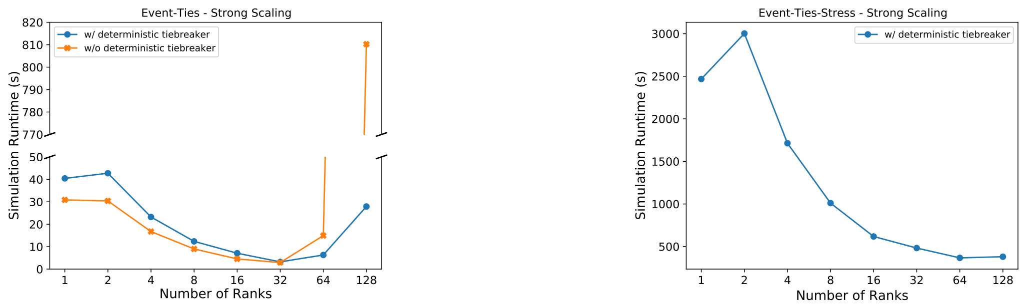
6 DISCUSSION

All simulations performed in this work were executed using at most 4 nodes of the AiMOS supercomputer at Rensselaer Polytechnic Institute's Center for Computational Innovations (Center for Computational Innovations, Rensselaer Polytechnic Institute). Each node contains two IBM Power 9 processors each with 20 cores with 3.15GHz clocks and 512GiB RAM. For simplicity of scaling experiments, we utilized a maximum of 32 processors per node. Inter-node communication is facilitated by an Infiniband EDR Fat Tree communication network.

6.1 Simulated Model Results

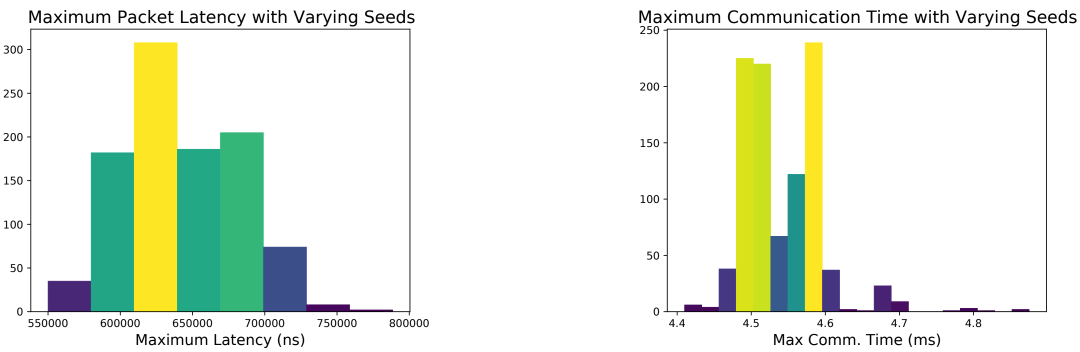
Generally the results of the strong scaling studies on the `PHOLD` model were typical. There was some minor overhead related to the generation of tie-breaking values for each new event and there was a slight performance reduction as a result, it became less noticeable at higher levels of parallelism.

The `Event-Ties` model, with its high degree of tied, zero-offset, events proved particularly difficult for ROSS to easily simulate without the tie-breaking feature. Shown in Figure 4a, as the simulation was executed with greater numbers of PEs, the runtime soared to abnormal levels. This was caused by the simulation engine's inability to make substantial progress as incomparable, simultaneous, events



(a) Event-Ties strong scaling study. (b) Event-Ties-Stress strong scaling study.

Figure 4: Strong scaling studies of models with and without the deterministic tie-breaking mechanism.



(a) Maximum end-to-end packet latencies histogram (b) Maximum application communication time histogram

Figure 5: Histograms of 1,000 communication network simulations with varying tie-breaking RNG seeds.

could not be committed to simulation history until GVT could progress and GVT could not effectively progress because at higher parallelism, the likelihood of out-of-order events becomes greater. Different configurations of ROSS simulation parameters could potentially improve its ability to handle larger degrees of zero-offset event ties but the trial-and-error process of tuning these parameters is not necessary if a tie-breaking mechanism was utilized.

Similarly, the `Event-Ties-Stress` model scaling study was not effectively simulated on ROSS without the tie-breaking mechanism at nearly any degree of parallelism and thus only the deterministic tie-breaking version results are displayed for this model in Figure 4b.

We also applied the capabilities of the tie-breaking mechanism to run 1,000 identical simulations of a LAMMPS communication workload across 2,048 nodes of a simulated 3,078 node CODES 1D-Dragonfly network – but each simulation featured a unique tie-breaking RNG seed. All tie-avoiding noise that is usually added to CODES event timestamps was removed, allowing for the occurrence of event-ties.

Aggregating all of the generated final measurement data, we can observe how different orderings of simultaneous events can have a drastic impact on final measurements and, consequently, any conclusions inferred from them. Figure 5 shows histograms of the maximum end-to-end packet latencies and simulated application communication time from these simulations. It is important to note that the RNG seeds that determine general LP behavior remained constant and that the variance in final simulation results depended only on the different simulation occurrences made possible by different orderings of simultaneous events.

If simultaneous events are avoided through artificial methods then ordering bias is introduced and fair sampling of all simulation occurrences becomes impossible. A fair tie-breaking mechanism makes a simulator tolerant of simultaneous events and allows for such studies to be performed.

Table 1: Subset of final results of the sequential $k = 1$, optimistic $k = 32$ and $k = 128$ runs for the `Event-Ties` model with and without the deterministic tie-breaker (TB).

Simulation	Net Events	LP1 Final Value
w/o TB: $k = 1$	39,190,528	35.0814
w/o TB: $k = 32$	39,190,528	34.4111
w/o TB: $k = 128$	39,190,528	34.4010
w/ TB: $k = 1$	39,190,528	34.7264
w/ TB: $k = 32$	39,190,528	34.7264
w/ TB: $k = 128$	39,190,528	34.7264

6.2 The Case for Determinism

All simulations in this work utilizing the deterministic tie-breaking feature were verified to exhibit the expected determinism. In contrast, simulations executed without it did not consistently generate a reproducible result.

One might argue that any additional overhead or performance impact is too great to warrant utilizing the feature. Simple models, such as `PHOLD`, have LP state that is not dependent on event ordering and event timings are already based on randomized values. Thus, even though event ties do occasionally occur in the simulated model, their ordering does not affect how many net events there are in the final simulation.

The same cannot be said for models sensitive to simultaneous event ordering. Table 1 shows extracted results from a few runs from the `Event-Ties` scaling results featured in Figure 4a. We observe small differences in the final results of LP state when there is no arbitration of simultaneous events – a lack of determinism. With the deterministic tie-breaker functionality enabled, however, determinism is restored and the parallel executions are identical to their sequential counterpart. The difference between the sequential runs with and without the tie-breaker is to be expected and is because the rules dictating order in the tie-breaker simulations also affect the order of events in the sequential version.

With the `Event-Ties` model – and *any* model whose generated events depend on LP state at time of processing – however, event ordering explicitly affects the final result. In the example shown in Table 1 it may appear that the executions without the tie-breaker were *close enough* but this is a product of the simplicity of the model and not indicative of what one could expect with other, more complicated, models. Table 1 also shows that simply looking at the number of net events alone is not sufficient for identifying determinism in a simulated model.

Even if a simulation completes faster without the overhead of a tie-breaking mechanism, the level of performance observed in a model with simultaneous events may simply be borrowed with determinism offered as collateral. The goal of parallel discrete event simulation is to recreate a semantically identical sequential simulation but distributed across different processing elements. If, from the general simulation engine's point of view, determinism in the parallel execution not assured, then it has, by definition, failed in this objective.

7 RELATED WORK

The topic of event simultaneity and ordering in parallel simulations has been discussed in various contexts over the span of decades (Rajaei and Ayani 1993; Ronngren and Liljenstam 1999; Kim, Seong, Kim, and Park 1997; Peschlow and Martini 2006; Peschlow and Martini 2007a; Peschlow and Martini 2007b; Lamport 1978; Wieland 1997). In (Lamport 1978), the topic of distributed logical and physical clocks, causal ordering conditions and happens-before relations is discussed but notes that the included theory makes no requirement about the ordering of incomparable or simultaneous events. The clocks discussed base their perception of time based on the perceived happens-before relations to other events.

Virtual time (Jefferson 1985) takes a slightly different approach by inferring happens-before relations from the encoded timestamps giving a simulated model the power to define when events happen in relation to each other. In (Schordan, Oppelstrup, Thomsen, and Glück 2020), the authors also extend ROSS' definition of virtual time to encode lower order bits to break event ties.

Much of this paper was inspired by the work performed in (Wieland 1997). The author makes arguments that given a simulation with n simultaneous events, then the *correct* final result is the mean of all $n!$ possible event orderings. Without some mechanism to effectively explore various possible orderings a correct final result is not achievable. In our work we provide a method that allows for random sampling of the possible event orderings.

That same year, the authors of (Kim, Seong, Kim, and Park 1997) proposed an event ordering mechanism for simultaneous events which also employed an extension to the basic timestamp for encoding event priority. This mechanism provided identical results between a sequential and parallel executed simulation and is similar in practice to our definition of biased random causal ordering.

The authors of (Ronngren and Liljenstam 1999) establish methods for breaking ties based on a logical clock interpretation of event causality or user defined priorities. They also briefly argue how different permutations of otherwise incomparable events should be considered and will inherently affect the outcomes of future events but do not elaborate on a solution to effectively explore this space. Similarly, in (Peschlow and Martini 2006; Peschlow and Martini 2007a; Peschlow and Martini 2007b) the authors note the importance of discovering other potential outcomes of simultaneous event orderings and propose a PDES branching mechanism tool that facilitates this.

8 CONCLUSION

In summary, we have proposed three solutions to the problem of maintaining a deterministic ordering of events in a parallel conservative or optimistically executed discrete event simulation given the possibility of n-way simultaneous events with and without zero-offset events. This solution leverages a deterministic pseudo-random number generator to encode values for use in breaking these n-way event ties into a specific, execution-consistent, ordering. This ordering can be unbiased and varied by changing the initial seed of the simulation's random number generators, which allows for further statistical analysis of execution results.

We evaluated the performance impact of the proposed solution and observed a marginal impact in a standard PDES benchmarking model. We also demonstrated that it can make it easier for simulations with a high degree of zero-offset events to make progress. Finally, we demonstrated how this feature can be useful for analysis in the area of network simulation.

REFERENCES

Carothers, C. D., D. Bauer, and S. Pearce. 2000. "ROSS: A High-performance, Low-memory, Modular Time Warp System". In *Proceedings of the Workshop on Parallel and Distributed Simulation*, 53–60. Bologna, Italy.

Carothers, C. D., D. Bauer, and S. Pearce. 2002. "ROSS: A High-performance, Low-memory, Modular Time Warp System". *Journal of Parallel and Distributed Computing* 62(11):1648–1669.

Center for Computational Innovations, Rensselaer Polytechnic Institute. "Artificial Intelligence Multiprocessing Optimized System (AiMOS)". https://cci.rpi.edu/aimos (accessed Nov 1, 2020).

Cope, J., N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross. 2011. "CODES: Enabling Co-design of Multilayer Exascale Storage Architectures". In *Proceedings of the Workshop on Emerging Supercomputing Technologies*, 6. Seattle, WA, USA.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest. 2001. *Introduction to Algorithms*. 3rd ed. Cambridge, MA, USA: MIT Press.

Fujimoto, R. M. 1990. "Parallel Discrete Event Simulation". *Communications of the ACM* 33(10):30–53.

Jefferson, D., B. Beckman, F. Wieland, L. Blume, and M. D. Loreto. 1987. "Time Warp Operating System". *ACM SIGOPS Operating Systems Review* 21(5):77–93.

Jefferson, D., B. Beckman, F. Wieland, L. Blume, M. D. Loreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J. Wedel, H. Younger, and S. Bellenot. 1987. "Distributed simulation and the time warp operating system". Technical Report OSTI 5639121, University of California, Los Angeles, CA, USA.

Jefferson, D., and H. Sowizral. 1985. "Fast concurrent simulation using the time warp mechanism". Technical Report ADA129431, Rand Corporation, Santa Monica, CA, USA.

Jefferson, D. R. 1985. "Virtual Time". *ACM Transactions on Programming Languages and Systems* 7(3):404–425.

Johnson, N. L., S. Kotz, and N. Balakrishnan. 1995. *Continuous Univariate Distributions, Volume 2*. 2nd ed. Hoboken, NJ, USA: John Wiley & Sons.

Kim, K. H., Y. R. Seong, T. G. Kim, and K. H. Park. 1997. "Ordering of Simultaneous Events in Distributed DEVS Simulation". *Simulation Practice and Theory* 5(3):253–268.

Lamport, L. 1978. "Time, Clocks, and the Ordering of Events in a Distributed System". *Communications of the ACM* 21(7):179–196.

McGlohon, N., and C. D. Carothers. 2021. "Unbiased Deterministic Total Ordering of Parallel Simulations with Simultaneous Events". ArXiV:2105.00069v1.

Peschlow, P., and P. Martini. 2006. "Towards an Efficient Branching Mechanism for Simultaneous Events in Distributed Simulation". In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, Volume 2006, 133.

Peschlow, P., and P. Martini. 2007a. "A Discrete-Event Simulation Tool for the Analysis of Simultaneous Events". In *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, 1–10. Nantes France.

Peschlow, P., and P. Martini. 2007b. "Efficient analysis of simultaneous events in distributed simulation". In *11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 244–251. Chania, Greece.

Rajaei, H., and R. Ayani. 1993. "Design Issues in Parallel Simulation Languages". *IEEE Design Test of Computers* 10(4):52–63.

Ronngren, R., and M. Liljenstam. 1999. "On Event Ordering in Parallel Discrete Event Simulation". In *Proceedings of the Workshop on Parallel and Distributed Simulation*, 38–45. Atlanta, Georgia, USA.

Schordan, M., T. Oppelstrup, M. K. Thomsen, and R. Glück. 2020. "Reversible Languages and Incremental State Saving in Optimistic Parallel Discrete Event Simulation". In *Reversible Computation: Extending Horizons of Computing*, 187–207. Cham, Switzerland: Springer International Publishing.

Wieland, F. 1997. "The Threshold of Event Simultaneity". In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, 56–59. Lockenhaus, Austria.

AUTHOR BIOGRAPHIES

NEIL MCGLOHON is a research scientist at Rensselaer Polytechnic Institute's Center for Computational Innovations (CCI). He received Ph.D and M.S. degrees in computer science from RPI in 2021 and 2016 respectively; in 2014 he received his B.S. in physics from the University of Oklahoma. His research interests include parallel and distributed computation and high performance computing network simulation including the exploration of the effects of network congestion on application performance through simulation. He acts as the primary maintainer for an open-source massively parallel discrete event simulation framework for simulating HPC communication networks to facilitate research collaborations with internal and external partners. nmcglo.rpi@gmail.com, https://nmcglo.com

CHRISTOPHER D. CAROTHERS is a faculty member in the Computer Science Department at Rensselaer Polytechnic Institute. He received the Ph.D., M.S., and B.S. from Georgia Institute of Technology in 1997, 1996, and 1991, respectively. Prior to joining RPI in 1998, he was a research scientist at the Georgia Institute of Technology. His research interests are focused on massively parallel computing which involve the creation of high-fidelity models of extreme-scale networks and computer systems. These models have executed using nearly 2,000,000 processing cores on the largest leadership class supercomputers in the world. Additionally, Professor Carothers serves as the Director for the Rensselaer Center for Computational Innovations (CCI). CCI is a partnership between Rensselaer and IBM. The center provides computational and storage resources to a diverse network of researchers, faculty, and students from Rensselaer, government laboratories, and companies across a number of science and engineering disciplines. The flagship supercomputer is an 8 peta-flop (PF) IBM AC922 hybrid CPU/GPU supercomputer named "AiMOS" which serves as the primary testbed for the IBM AI Hardware Center. chris.carothers@gmail.com, https://www.cs.rpi.edu/~chrisc/