Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

OPTIMIZING SIMULATION ON SHARED-MEMORY PLATFORMS: THE SMART CITIES CASE

Mauro Ianni, Romolo Marotta
Davide Cingolani, Alessandro Pellegrini

Sapienza, University of Rome
Rome, ITALY

Francesco Quaglia

University of Rome “Tor Vergata”
Rome, ITALY

ABSTRACT

Modern advancements in computing architectures have been accompanied by new emergent paradigms to run Parallel Discrete Event Simulation models efficiently. Indeed, many new paradigms to effectively use the available underlying hardware have been proposed in the literature. Among these, the Share-Everything paradigm tackles massively-parallel shared-memory machines, in order to support speculative simulation by taking into account the limits and benefits related to this family of architectures. Previous results have shown how this paradigm outperforms traditional speculative strategies (such as data-separated Time Warp systems) whenever the granularity of executed events is small. In this paper, we show performance implications of this simulation-engine organization when the simulation models have a variable granularity. To this end, we have selected a traffic model, tailored for smart cities-oriented simulation. Our assessment illustrates the effects of the various tuning parameters related to the approach, opening to a higher understanding of this innovative paradigm.

1 INTRODUCTION

Parallel Discrete Event Simulation (PDES) (Fujimoto 2001) is a powerful method to support large-scale and complex discrete-event simulation systems. This paradigm has seen an enormous interest from the research community in the past 30 years, due to the fact that it is possible to decouple, to a very large extent, the actual simulation model and the runtime environment in charge of supporting its execution.

This has allowed PDES to survive many changes in the underlying computing infrastructure. As an example, the recent trend towards exascale computing and the advent of the power wall (Sutter 2005) and the memory wall (McKee 2004) have induced profound changes in how computing architectures are organized, in order to deliver an always increasing computing power. To overcome the power wall, the industry has started producing multicore chips, which have led to the adoption of massively-parallel single-node computing systems, with a non-minimal core count and with or without coprocessors. The continuous increase in core count has led to the adoption of new memory organizations in order to reduce contention and synchronization needs across cache controllers, leading to the adoption of Non-Uniform Memory Access (NUMA) machines. In contrast to Uniform Memory Access (UMA) machines, NUMA ones have memory banks divided in different nodes, which provide a different access latency when a memory access is issued by different cores—they can be farther or closer to a certain subset of physical memory.

PDES has survived all these drastic changes in the underlying hardware. This is mainly due to the fact that the separation between the model’s code and the runtime environment allows to keep the effort made into model development, while re-adapting (or re-designing) the underlying runtime environment allows to benefit from new hardware organization.

This has been particularly the case for a well-studied branch of PDES, namely *speculative PDES*, which tries to capture the degree of parallelism of simulation models by relying on optimistic synchronization protocols, such as Time Warp (Jefferson 1985), both on shared-memory and distributed parallel systems. In Time Warp-based optimistic PDES, events are executed independently of their safety, in the hope that "nothing will go wrong". An a-posteriori detection system is able to detect the arise of causality violations, i.e. the receipt of straggler events which cannot be correctly executed anymore because they refer to a previous logical time with respect to the last-executed event. Correctness of the simulation is ensured by the ability of such systems to restore a previous (consistent) simulation state, from which to restart the execution of events. This way of dealing with causality violations has been shown to be quite independent of the lookahead of the simulation, network latency, and other parameters of the model with respect to other optimistic synchronization protocols such as (Nicol, Michael, and Inouye 1989), making it a fervent research field even after more than 30 years from its initial proposal.

With respect to massively-parallel shared-memory machines, a recent research trend has started to study the so-called *share-everything* processing paradigm. The seminal papers in (Hay and Wilsey 2015; Santini, Ianni, Pellegrini, and Quaglia 2015; Marotta, Ianni, Pellegrini, and Quaglia 2016) show how it is possible, on a multicore shared memory machine, to achieve a non-minimal speedup with respect to more traditional paradigms, when dropping fundamental constraints such as the complete data separation across Logical Processes (LPs). In particular, the work in (Marotta, Ianni, Pellegrini, and Quaglia 2016) has shown that it is possible to rely on non-blocking algorithms (Herlihy and Shavit 2011) to let concurrent threads access a single shared priority queue. At the same time, the work in (Santini, Ianni, Pellegrini, and Quaglia 2015) has highlighted the possibility to rely on a different runtime controller to determine the consistency of events processed speculatively. While the latter work relies on hardware transactional memory to assess the safety of events, this has paved the way to other proposals which rely on pure software solutions to achieve the same result.

All this body of work has given rise to the share-everything paradigm, which has been recently specified in (Ianni, Marotta, Cingolani, Pellegrini, and Quaglia 2018). This work has shown how it is possible, on a shared-memory machine, to significantly reduce the number of events that are undone in the speculative simulation portion. This has the effect of providing a significantly increased efficiency—under certain circumstances it is as high as 100%—which in turn offers a non-negligible speedup with respect to traditional PDES systems. Moreover, as we will show later, the share-everything PDES paradigm bridges the gap between conservative and speculative simulation, thus providing a new runtime paradigm which allows to exploit the benefits of both execution models. Nevertheless, the assumption which has driven the construction of the share-everything simulation paradigm is that traditional PDES systems are inefficient whenever they have to deal with very fine-grain simulation models. In particular, if the average duration of events is below tens of microseconds, then traditional rollback mechanisms have an overhead so large that it is not paid off.

The goal of this paper is to provide a comprehensive study of the share-everything simulation paradigm, on massively parallel shared-memory machines, by explicitly violating the assumption which drove the definition of the paradigm. Indeed, we rely on a simulation benchmark for smart cities, which offers microsimulation capabilities of traffic jams at the details of a single car in a street. Given the high amount of data to be processed, the duration of events in this case is much larger, on the order of milliseconds. The ultimate goal of this work, therefore, is to study what is the tradeoff between the share-everything and traditional (e.g., data separation-based) PDES systems, with the aim of showing in what cases one approach could be better than the other. This is therefore a fundamental study having in mind the possibility to devise simulation systems which rely on both approaches in a synergistic way, and are capable of switching at runtime from one approach to the other, in an autonomic way.

The remainder of this paper is structured as follows. Section 2 illustrates the rationale behind the share-everything simulation paradigm. In Section 3, we discuss the technical details behind the reference share-everything runtime environment which has been used for the experimental assessment. The comprehensive experimental results are then presented in Section 4. Related work is discussed in Section 5.

2 THE SHARE-EVERYTHING PDES PARADIGM

The share-everything PDES paradigm aims at guaranteeing scalability along two orthogonal dimensions in a combined manner, namely: i) wall-clock time coordination across the different Worker Threads (WTs) which carry on the execution of events, and ii) virtual-time coordination across different simulation objects. As for point i), this is achieved by relying on non-blocking algorithms (Marotta, Ianni, Pellegrini, and Quaglia 2017; Ianni, Marotta, Pellegrini, and Quaglia 2017) which allow concurrent access on a shared event pool, without the need to rely on, e.g., spinlock-protected critical sections. These non-blocking algorithms implement an access strategy to avoid collisions across threads in the access both to the state of the simulation objects—this allow to reduce contention on private simulation states—and to simulation platform's metadata. Technically, the share-everything PDES paradigm is based on the following advanced concurrency ideas:

- non-blocking management of a fully-shared pending-event set that contains both schedule-committed events (those produced by the execution of other events that have been detected to be safe and causally consistent) and non schedule-committed ones (those that are the result of speculative, not yet committed, processing actions), which might need to be (logically) canceled;
- non-blocking dispatching of events to be processed by WTs in order to avoid collision on a same simulation object and so that causal consistency is detected on-the-fly. In this way, it is possible to optimize the way events are actually processed (in terms of configuration of event-undo support).

In this sense, the share-everything speculative PDES paradigm allows to combine on a per-single-event basis both conservative and speculative processing techniques. The generality of this paradigm is also such that restoring a previous non-consistent state can be supported by different methodologies, e.g. traditional state saving and restore (Ianni, Marotta, Cingolani, Pellegrini, and Quaglia 2018), reverse scrubbing (Cingolani, Ianni, Pellegrini, and Quaglia 2016), or transactional memory (Santini, Ianni, Pellegrini, and Quaglia 2015).

Moreover, this paradigm can cope with hard-workload scenarios where there are (sudden) skews in the distribution of the events across simulation objects along virtual time. These skews possibly create relatively short bursts of events to be processed at a subset of the objects, while other objects have no (or few) events to be processed along that same virtual time window. In these scenarios traditional PDES-oriented load balancing approaches, based on medium-term binding between objects and threads, have scarce capability to react to the sudden unbalance that may materialize, which can lead to an increase of the likelihood of wasted computation in case of speculative processing. The share-everything paradigm considers events as fully-shared workload units, thus being able to concentrate the computing power on any burst of events that materializes among subsets of objects. Indeed, WTs can take care of processing any event in these bursts, thus contributing to promptly advance the currently hot portions of the simulation model.

In a way similar to traditional data-partitioned PDES, the share-everything PDES paradigm targets models which are partitioned into different simulation objects, the execution of which is carried out by LPs. LPs are sequential entities which are dispatched when a Worker Thread (WT), triggers the execution of the handler of an event destined to it. Nevertheless, differently from traditional parallel PDES systems, share-everything PDES allows any WT in the system, at any time, to schedule an event destined to any LP in the system. To this end, differently from the traditional speculative Time Warp organization (Jefferson 1985), a single unique event pool is used, to store and extract events targeting all the LPs.

3 REFERENCE RUNTIME ENVIRONMENT ORGANIZATION

For a thorough discussion of the architectural details of a reference share-everything simulation platform, we refer the reader to the work in (Ianni, Marotta, Cingolani, Pellegrini, and Quaglia 2018). We discuss here the most fundamental aspects of the reference implementation, for the sake of completeness. There are two main data structures which support share-everything simulation:

- a set of *LP Control Blocks* (LPCBs), used to keep metadata representing the system-level view of the advancement of the LP in simulation time;
- a set of events, maintained into the aforementioned fully-shared unique pool which we refer to as *Scheduling Queue* (SQ).

The SQ is such that the events it contains at any time can belong to three different categories: *unprocessed events*, i.e. events associated to a future part of the speculative execution of the simulation; *uncommitted events*, i.e. events which are already processed but are not yet associated with a stable portion of the simulation trajectory; *canceled events*, namely events which have been annihilated by the reception of an antimessage, but are still to be removed from the queue—SQ is subject to *lazy cancellation* of annihilated events for performance reasons.

At first approximation, the main execution loop carried out by all the WTs consists in: i) fetching some event to be processed from the SQ; ii) performing a rollback of the target LP, if required; iii) executing the event; iv) updating the LPCB; v) inserting newly-generated events into the SQ.

As for point i), the fetch operation is based on a try-lock on the target LP. Hence, no WT will ever fetch an event bound to an LP that is currently locked for execution by some other WT. Overall, no mutual block among WTs will ever occur in the attempt to access the same LP, since the WT that will experience a failure of its try-lock operation will simply go ahead scanning the event pool in order to take an event destined to some other LP. This also guarantees isolation of WTs' accesses to a given LPCB and to the corresponding simulation object state, in both forward and rollback mode. All other operations to access the SQ rely on non-blocking algorithms.

Each LPCB keeps metadata needed by the simulation engine to detect any relevant runtime condition related to the LP, including its involvement in causality errors. Among the fundamental metadata used to support share-everything simulation we find:

- `bound`, namely a pointer to the last correctly executed event in the not-yet-committed part of the simulation. This pointer allows to perform a fast extraction of the next event to be processed by any WT in the system, and to detect whether a causal inconsistency is being faced due to a straggler message being received.
- `epoch`, which keeps track of the total number of rollback operations the LP has experienced. This value is used in order to implement lazy cancellation of events annihilated by antimessages. Indeed, events are tagged with the epoch number in which they were generated: if a LP is living in a new epoch, after a rollback, this information implicitly tells what messages should be discarded due to the fact that the sender LP is now following a different path in the simulation trajectory.
- `local_queue`, i.e. a parallel view on the events kept by SQ which allows to navigate through events which have already been processed. This view allows to effectively perform state reconstruction, which is supported in the share-everything PDES paradigm by means of checkpointing and coasting forward.
- An LP lock, which is used to implement the try-lock strategy described before.

Given that, thanks to the try-lock mechanism, two events destined to the same LP cannot be concurrently processed by WTs, we can exploit the lookahead (LA) of the simulation model to compute safety of whatever event to be processed (or being processed) according to the following expression:

$$is\_safe(e) = (e.\texttt{ts} \in [GVT, GVT + LA) \;\wedge\; \nexists\, e' :\; e.\texttt{lp} = e'.\texttt{lp} \;\wedge\; e'.\texttt{ts} < e.\texttt{ts}) \qquad (1)$$

Equation (1) tells that if an event *e* is not safe and it is speculatively processed, its execution might be undone due to the arrival of a straggler message destined to the same LP. However, in such a scenario, the event *e* could be still *valid*, meaning that it must appear along some timeline of the destination LP. On the other hand, if the event was generated by some other event that is undone, the former becomes *invalid*. In fact, it should never appear in the final timeline of the destination LP—in classical Time Warp these are events canceled by their corresponding anti-events.

As for the safety, a particular event *e* is detected to be invalid at a given point in wall-clock time by checking whether its generating event *p* is also currently valid and if the execution of *p* that generated *e* has not been undone.

The SQ is a conflict-resilient lock-free priority queue (Marotta, Ianni, Pellegrini, and Quaglia 2017) that sorts events on the basis of their timestamps. At a logical level, such a queue can be abstracted as a generic non-blocking ordered linked list like the one proposed by Harris (Harris 2001)—although being much more efficient thanks to its multi-bucket organization leading to amortized constant-time access. SQ exposes an API composed of several different functions:

- ENQUEUE, which allows to store a new event in timestamp order.
- GETMIN, which retrieves a pointer to the event with the smallest timestamp which is still linked to the queue
- GETNEXT, which retrieves the pointer to the event which immediately follows in virtual time the one identified by its input argument.
- UNLINK, which allows to disconnect any node from the queue.

The share-everything PDES platform implements a FETCH operation relying on this API. It returns in a non-blocking mode a to-be-processed event associated with some LP not currently locked by any WT. Indeed, before returning an event *e*, a WT executing a FETCH executes a try-lock operation on the target LP. If this operation fails, the WT slides to the next event in the queue via a GETNEXT. This is done until a try-lock on the LP targeted by some event, encountered along the queue, executes successfully.

In the main simulation loop, a call to the FETCH procedure is executed to retrieve from the SQ an event to be handled (processed or undone/retracted), which is destined to an LP not currently in charge of another WT. The FETCH procedure returns to the caller WT a pointer to the event to be handled, and the indication of whether the event is safe and/or valid.

Regardless the retrieved event's current state, the WT checks whether its timestamp is smaller than the LP's LVT by checking the event pointed by `bound`. In the positive case, a ROLLBACK is triggered in order to bring back the LP's state to a consistent snapshot.

Event processing starts by updating the event's epoch with that of the LP, to represent its current incarnation within the LP's timeline. New events possibly produced during the execution are stored in a local buffer, flashed to the SQ at the end of the event's execution. At this point, to complete event processing, the event is atomically set as executed, and the LP's bound is updated so as to point to the current event.

At the end of the main loop, housekeeping operations take place. Their goal is to reclaim memory associated with no longer needed event buffers and checkpoints. Similarly, at this point, simulation termination might take place, depending on the actual termination condition which is specified by the model (e.g., related to a certain simulation time being reached, or some property of the simulation state being met).

4 EXPERIMENTAL ASSESSMENT

4.1 The Benchmark for Traffic in Smart Cities

As mentioned before, the goal of the experimental assessment provided in this paper is to show the behaviour of the share-everything PDES paradigm when dealing with simulation models which require a non-minimal amount of CPU time in order to process events, and which rely on imbalanced message-exchange topology. Indeed, the share-everything paradigm has been devised with small granularity events in mind. Nevertheless, the effectiveness of this paradigm in previous applications shown in the literature, such as in (Ianni, Marotta, Cingolani, Pellegrini, and Quaglia 2018) suggested that it can be effectively exploited also in scenarios which a much coarser grain event granularity.

To perform this specific assessment, we have relied on the *traffic* simulation model. This is a vehicular mobility model which has already been extensively used to assess the performance of traditional PDES systems—see, e.g., (Vitali, Pellegrini, and Quaglia 2012). This simulation model has been developed to perform simulations for infrastructure planning in the context of smart cities. This specific kind of what-if analysis is such that it allows to determine beforehand whether a certain part of a road network is suitable to manage large to intensive amounts of traffic.

The traffic model simulates any road network topology at the granularity of a single car. The topology of the road network to be studied can be specified in a configuration file as a generic graph. Nodes represent junctions in the network, while edges represent actual roads.

In this experimental assessment, we have configured the traffic model so as to simulate the entire Italian highway system (islands excluded, to have a fully-connected graph). Every node is described in terms of car interarrival time and car leaving probability (therefore describing its load), while edges are described in terms of their length. Every LP handles the simulation of a node or a portion of a segment, the length of which depends on the total highway's length and the number of available LPs.

Cars enter the system according to an Erlang probability distribution, with a mean interarrival time specified (for each node) in the topology configuration file. They can join the highway starting from cities/junctions only, and are later directed towards highway segments with a uniform probability. Whenever a car is received, it is enqueued in the LP's list of traversing cars, and its speed (for the particular LP it is entering in) is determined according to a Gaussian probability distribution, the mean and the variance of which is specified at startup time. Then, the model computes the time the car will need to traverse the road segment, adding traffic slowdowns which are again computed according to a Gaussian distribution. In particular, the probability of finding a traffic jam is a function of the number of cars which are currently passing through the road segment.

Accidents are derived according to a probability function as well. In particular, they are more likely to occur when the amount of cars traversing an LP is about half of the cars which can be hosted altogether. In fact, if few cars are in, accidents are less frequent, while if there are many, the traffic factor produces a speed slowdown, entailing the probability of an accident to occur to be reduced. Therefore, the model discretizes a Normal distribution, computing the Cumulative Density Function in a contour defined as $\textit{cars in the node} \pm \frac{1}{2}$, having as the mean half of the total number of cars which are at the current moment in the system, and as variance a factor which can be specified at startup. The total number of cars which can be hosted by an LP is computed according to the actual length of the simulated road, which is determined when the model is initialized. When an accident occurs, the cars are not allowed to leave the LP, until the road is freed. The duration of an accident phase is determined according to a Gaussian distribution, again parameterizable at startup.

4.2 Experimental Results

All the tests shown in this paper have been run on a 32-core HP ProLiant machine running Linux (kernel 3.2) equipped with 64 GB of RAM. The number of WTs running within the PDES platform has been varied from 1 to 32, just in order to perform a scalability study. The used implementation of the used share-everything simulation engine can be found online at https://github.com/HPDCS/USE.

The topology which we have used to configure the traffic model represents the whole Italian highway network (islands excluded) on less than 200 LPs. While this could appear as a small simulation, we note that the small number of LPs involved in the simulation is an actual worst-case scenario for the share-everything PDES paradigm. Indeed, this configuration shows a very high degree of parallelism, which therefore exacerbates the likelihood that a try-lock operation on an LP will fail. In this sense, the probability that a WT has to scan through the pending-event set (by means of the GetNext operation) to pick an event which can be actually executed is non-minimal. Average speed has been set to 110 Km/h, with a variance of 20 Km/h, and accident durations have been set to 1 hour, with 30 minutes variance. The topology and the variance in car speed is such that the overall amount of work destined to some LPs is much higher than others, in terms of distribution of events to be processed along the pending-event set.

Moreover, each LP is in charge of executing coarse-grain events. Indeed, the simulation model has a very precise accuracy, such that each single car is explicitly modeled and managed. In this sense, each LP, when running an event, has to deal with a non-minimal amount of data structures. Indeed, the statistics associated with each car (e.g., its speed, whether it is involved in an accident, whether it is stopping in a filling station or at a lay-by) are recomputed while processing any event in the system.

Some LPs have to deal with thousands of cars, while others have to deal with just tens. Overall, this creates an imbalanced execution time of events which could generate skews of events, that can be effectively handled by the share-everything PDES paradigm. At the same time, events have an average granularity of up to 2 milliseconds, while the share-everything paradigm has been initially conceived for a granularity of the order of tens of microseconds. Therefore, this setup allows us to stress test the share-everything PDES paradigm.

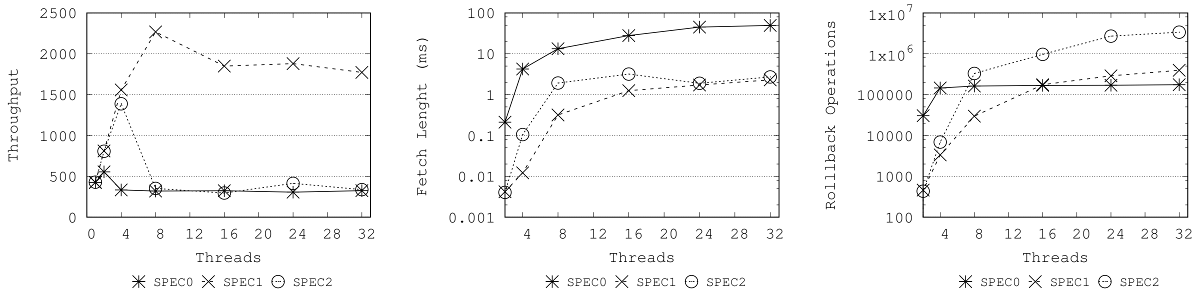
The topology which has been used to configure the traffic benchmark shows runtime dynamics of the model which can be regarded of a bad case for share-everything PDES systems as: i) the number of cross-LP events is very high (almost half of the overall generated events is a cross-LP one); ii) these cross-LP events are simultaneous events, stressing the tie-breaking facilities of the share-everything PDES systems; iii) the event granularity is extremely variable, ranging from tens of nanoseconds to 2 milliseconds.

In our assessment we have conducted two families of experiments. One relies on 137 LPs, with the LPs representing an edge in the road graph which are associated with different road lengths (ranging from tens to hundreds of kilometers). Due to the single car granularity of the model, this configuration is such that LPs managing longer road segments have a larger amount of work to be carried out during the execution of simulation events, thus making it a quite imbalanced scenario—we refer to this setup as Imbalanced Topology (IBT) in the plots. The second family of experiments still simulates the whole road topology, yet longer roads are split over multiple LPs, in such a way that each LP manages a similarly long piece of road. This creates a more balanced setup—we refer to it as Balanced Topology (BT) in the plots. Both families of experiments are such that each car in the system is configured so as to travel an average distance of around 100 Km. This ensures that the amount of cross-LP messages is non minimal, due to the nature of the topology in which the average length of a segment is of the order of 25 Km.

For both families of experiments we have varied the interarrival time τ of cars entering the highway to an average value of 3 to 10 seconds of simulation time. This is compliant with scenarios of medium to high load of the highway national system. The overall average event granularity for the different configurations are reported in Table 4.2, which shows that we range from a medium to a large granularity. For each family of experiments we report the overall simulation throughput, the duration of the fetch phase (which is an indication of the contention on the platform level metadata), and the total number of rollback operations. Three different configurations of the share-everything PDES platform are evaluated. They are associated with different levels of speculation implemented on top of the shared queue, referred to as SPEC0, SPEC1, and SPEC2. In SPEC0, we significantly limit the degree of speculation by avoiding to pick for execution events which belong to LPs which are either blocked, or which have processed events which are not yet safe.

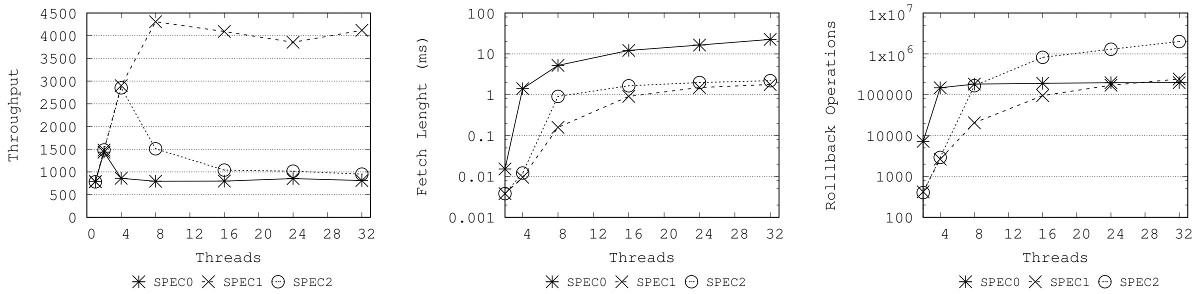
	IBT	**BT**
$\tau = 3$	1.5 ms	2 ms
$\tau = 10$	0.1 ms	0.15 ms

Table 1: Average Event Duration.



(a) (b) (c)

Figure 1: Experimental Results: $\tau = 3$, IBT.

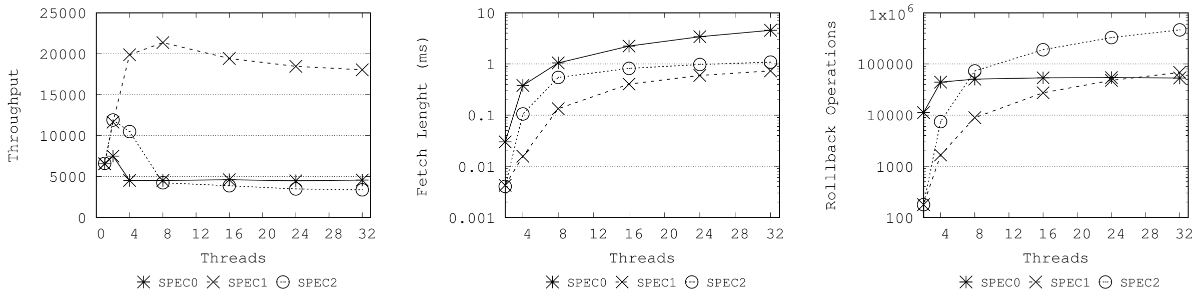


(a) (b) (c)

Figure 2: Experimental Results: $\tau = 3$, BT.

On the other hand, in SPEC1, once the try-lock operation on a certain LP fails, the WT will never attempt to perform such operation on different events bound to the same LP until an event is executed. This reduces the probability that an out of order event is executed at an LP. Finally, SPEC2 always executes the try-lock operation on all LPs, regardless of what happened before, while scanning SQ. This increases the likelihood that an event is picked for execution early, but increases the probability that it is causally-dependent on a previous event which has not yet been processed.

Experimental results for the configuration with $\tau = 3$ are shown in Figures 1 and 2. By the results, we observe that the speculativity level SPEC2 does not pay off. This is related to the fact that the number of incorrectly executed events is higher, producing a total number of rollback operations which is almost 10 times larger. The rollback operation in the used share-everything platform requires taking a large number of locks on the involved LPs. In turn, this entails that the number of try-lock operations which fail at different concurrent WTs is non-minimal. This generates a longer duration of the fetch phase. We note that if the fetch phase is longer, some WTs are picking event much farther in the simulation time. This increases again the likelihood that the picked event is not causally consistent, thus generating a waterfall effect in the rollback pattern. The SPEC1 configuration, on the other hand, is more tailored to selecting for execution causally-consistent events. This strategy shows a simulation throughput which grows up to

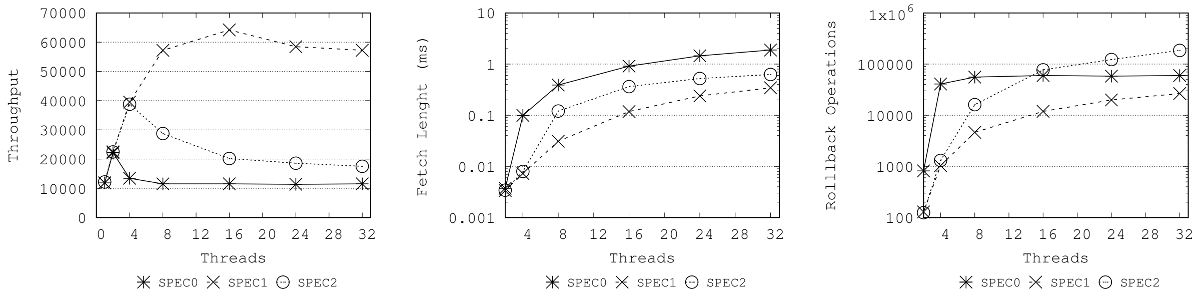


(a)

(b)

(c)

Figure 3: Experimental Results: $\tau = 10$, IBT.



(a)

(b)

(c)

Figure 4: Experimental Results: $\tau = 10$, BT.

8 concurrent threads. When the number of threads is higher than 8, due to the reduced number of LPs involved in the simulation, the degree of parallelism of the simulation run is higher. In this case, the incidence of rollbacks increases, as in traditional PDES systems. Nevertheless, the try-lock-based strategy is such that the overall throughput of the simulation is not degraded significantly. By comparing the number in Table 4.2, we note that the IBT configuration with $\tau = 3$ is the one with the higher event granularity. Therefore, the experimental results in Figure 1 confirm the intuition that the share-everything simulation paradigm can be effective also when dealing with events whose duration is non-negligible. The results in Figure 3 are an additional confirmation of the effectiveness of the share-everything paradigm when events have a small-to-medium duration—here, the average granularity is 0.05 ms. SPEC0 suffers from problems of both SPEC1 and SPEC2, and this is why it always exhibits a bad performance.

The results in Figures 3 and 4 are related to the balanced case. They again confirm the rationale behind the share everything paradigm. Indeed, when the event granularity is even higher (1.5 ms), a too high degree of parallelism (e.g., when running 137 LPs on top of 32 WTs), starts to produce a reduction in the simulation throughput, in all SPEC configurations. This is due to the fact that when events are very large, the fetch duration phase becomes the bottleneck of the system. We remind that the fetch phase is one of the fundamental building blocks to devise a non-blocking access to the shared event pool. This execution pattern creates once again a waterfall effect on the number of rollback operations which are executed.

5 RELATED WORK

The share-everything simulation paradigm targets shared-memory machines. This topic has been addressed by several works to meet various objectives. In (Swenson and Riley 2012; Vitali, Pellegrini, and Quaglia 2012; Wang, Jagtap, Abu-Ghazaleh, and Ponomarev 2014; Wang, Abu-Ghazaleh, and Ponomarev 2015) the authors provide solutions to reshuffle traditional-style PDES systems, making them more suited for shared-memory platforms. Few solutions optimize the architecture of the communication facilities across the threads. Other solutions take advantage of the possibility for any thread to promptly access the state of any simulation object and of its event queue when a rebinding between objects and threads is needed—for load balancing. In some cases, interference from external workload is also considered in the rebinding. Conversely, share-everything considers single events as work units to dispatch to any WT.

The works in (Chen, Lu, Yao, Peng, and Wu 2011; Pellegrini, Peluso, Quaglia, and Vitali 2016; Pellegrini and Quaglia 2014) rely on shared memory for to allow information sharing across simulation objects. This is done by means of transactional memory, software instrumentation or operating system facilities. However, these proposals are still bound to the traditional PDES paradigm. In fact, they have been integrated into environments that still rely on workload partitioning across WTs, rather than fine-grain sharing of individual work units.

Providing event-pool data structures enabling concurrent accesses has been addressed in (Ayani 1990), which proposes an approach based on fine-grain locking of a sub-portion of the data structure upon performing an operation. However, the intrinsic scalability limitations of locking still lead this proposal to be not suited for large levels of parallelism, as also shown in (Rönngren and Ayani 1997). Share-everything relies on non-blocking algorithms, which have been shown to be much more prone to scalability.

Non-blocking operations on event pools have also been studied in (Gupta and Wilsey 2014), which presents a variation of the Ladder Queue where the elements are at any time bound to the correct bucket, which is an unordered list. The extraction from an unordered bucket returns the first available element, which does not necessarily correspond to the one with the minimum timestamp. This proposal is intrinsically tailored for PDES systems relying on speculative processing, where unordered extractions leading to causal inconsistencies within the simulation model trajectory are reversed (in terms of their effects on the simulation model trajectory) via rollback mechanisms. In our proposal we guarantee the ordering of the events in the shared pool, which allows us to put in place the smart combination of conservative (say safe) and speculative processing at the level of each individual event—also thanks to the explicit exploitation of the lookahead in the simulation model—thus enabling the optimization of the rollback support, an aspect that is not considered in (Gupta and Wilsey 2014).

The recent proposal in (Hay and Wilsey 2015) explores the idea of managing concurrent accesses to a shared pool by relying on Hardware Transactional Memory (HTM) support. Insertions and extractions are performed as HTM-based transactions, hence in non-blocking mode. However, the level of scalability of this approach is limited by the level of parallelism in the underlying HTM-equipped machine, which nowadays is relatively small. Also, HTM-based transactions can abort for several reasons, not necessarily related to conflicting concurrent accesses to a same portion of the data structure. As an example, they can abort because of conflicting accesses to the same cache line by multiple CPU-cores, which might be adverse to PDES models with, e.g., very large event pools. Share-everything PDES does not require special hardware support, thus fully eliminating the secondary effects caused by, e.g., HTM limitations on the abort rate of the operations.

6 CONCLUSIONS AND FUTURE WORK

We have conducted an experimental assessment of a simulation platform built according to the share-everything PDES paradigm devised for massively-parallel simulation systems. We have used the traffic simulation model, which is tailored to car-accurate simulations for smart cities environments.

The experimental results have shown that share-everything PDES can be effective also when dealing with events whose duration is non-minimal. As soon as the event's granularity becomes too large, the bottleneck has been identified in the fetch phase of the considered scheduling approach. These results allow to settle future work directions towards the design and implementation of different scheduling policies for share-everything PDES systems, and give an indication of what could be the trade-off for the integration of traditional PDES systems and share-everything systems.

REFERENCES

Ayani, R. 1990. “LR-Algorithm: concurrent operations on priority queues”. In *Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing*, SPDP, 22–25. Dallas, TX, USA: IEEE Computer Society.

Chen, L.-l., Y.-s. Lu, Y.-P. Yao, S.-l. Peng, and L.-d. Wu. 2011. “A Well-Balanced Time Warp System on Multi-Core Environments”. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, PADS, 1–9: IEEE Computer Society.

Cingolani, D., M. Ianni, A. Pellegrini, and F. Quaglia. 2016. “Mixing Hardware and Software Reversibility for Speculative Parallel Discrete Event Simulation”. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Volume 9720, 137–152.

Fujimoto, R. M. 2001. “Parallel simulation: parallel and distributed simulation systems”. In *Proceedings of the 33nd conference on Winter simulation*, WSC, 147–157: IEEE Computer Society.

Gupta, S., and P. A. Wilsey. 2014. “Lock-free pending event set management in Time Warp”. In *Proceedings of the 2014 ACM/SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS, 15–26: ACM Press.

Harris, T. 2001. “A Pragmatic Implementation of Non-blocking Linked-lists”. In *Distributed Computing*, edited by J. Welch, Volume 2180, 300–314. Springer Berlin / Heidelberg.

Hay, J., and P. A. Wilsey. 2015. “Experiments with hardware-based transactional memory in parallel simulation”. In *Proceedings of the 2015 ACM/SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS, 75–86: ACM Press.

Herlihy, M., and N. Shavit. 2011. “On the nature of progress”. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Volume 7109 LNCS, 313–328.

Ianni, M., R. Marotta, D. Cingolani, A. Pellegrini, and F. Quaglia. 2018. “The ultimate share-everything PDES System”. In *2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation.*

Ianni, M., R. Marotta, A. Pellegrini, and F. Quaglia. 2017. “Towards a fully non-blocking share-everything PDES platform”. In *Proceedings - 2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2017*, Volume 2017-Janua.

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

Marotta, R., M. Ianni, A. Pellegrini, and F. Quaglia. 2016. “A Non-Blocking Priority Queue for the Pending Event Set”. In *Proceedings of the 9th ICST Conference of Simulation Tools and Techniques*, SIMUTools: ICST.

Marotta, R., M. Ianni, A. Pellegrini, and F. Quaglia. 2017. “A conflict-resilient lock-free calendar queue for scalable share-everything PDES platforms”. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*.

McKee, S. a. 2004. “Reflections on the memory wall”. *Proceedings of the first conference on computing frontiers on Computing frontiers - CF’04*:162.

Nicol, D. M., C. C. Michael, and P. Inouye. 1989. “Efficient aggregation of multiple PLs in distributed memory parallel simulations”. In *Proceedings of the 21st conference on Winter simulation - WSC ’89*, 680–685. New York, New York, USA: ACM Press.

Pellegrini, A., S. Peluso, F. Quaglia, and R. Vitali. 2016, apr. “Transparent speculative parallelization of discrete event simulation applications using global variables”. *International Journal of Parallel Programming*.

Pellegrini, A., and F. Quaglia. 2014. “Transparent Multi-Core Speculative Parallelization of DES Models with Event and Cross-State Dependencies”. In *Proceedings of the 2014 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS, 105–116: ACM.

Rönngren, R., and R. Ayani. 1997. “A Comparative Study of Parallel and Sequential Priority Queue Algorithms.”. *ACM Transactions on Modeling and Computer Simulation* 7(2):157–209.

Santini, E., M. Ianni, A. Pellegrini, and F. Quaglia. 2015. “HTM Based Speculative Parallel Discrete Event Simulation of Very Fine Grain Models”. In *Proceedings of the 22nd International Conference on High Performance Computing*, HiPC: IEEE Computer Society.

Sutter, H. 2005. “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software”. *Dr. Dobb’s Journal* 30(3):202–210.

Swenson, B. P., and G. F. Riley. 2012. “A New Approach to Zero-Copy Message Passing with Reversible Memory Allocation in Multi-core Architectures.”. In *PADS*, 44–52.

Vitali, R., A. Pellegrini, and F. Quaglia. 2012. “Towards Symmetric Multi-threaded Optimistic Simulation Kernels”. In *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, PADS, 211–220: IEEE.

Wang, J., N. Abu-Ghazaleh, and D. Ponomarev. 2015. “AIR: Application-Level Interference Resilience for PDES on Multicore Systems”. *ACM Trans. Model. Comput. Simul.* 25(3):19:1—-19:25.

Wang, J., D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev. 2014. “Parallel discrete event simulation for multi-core systems: Analysis and optimization”. *IEEE Transactions on Parallel and Distributed Systems* 25(6):1574–1584.

AUTHOR BIOGRAPHIES

MAURO IANNI is a PhD Student at Sapienza, University of Rome. He focuses on methodologies and techniques to ensure the correctness of operations in concurrent environment, applications to support data processing on massively parallel environments, theory of computability, and the design and the development of high-performance synchronization algorithms. His e-mail address is mianni@diag.uniroma1.it.

ROMOLO MAROTTA is enrolled in a Doctoral Program at Sapienza, University of Rome. His research targets the development of new computability models for highly-parallel systems. He studies as well non-blocking synchronization algorithms of general applicability to data processing and analytics. His e-mail address is marotta@diag.uniroma1.it.

DAVIDE CINGOLANI is a PhD Student at Sapienza, University of Rome. His main research activities target transparent software reversibility, both from a theoretical/semantic point of view, and from the perspective of practical realization of middleware to enforce reversible execution of generic programs. His e-mail address is cingolani@diag.uniroma1.it.

ALESSANDRO PELLEGRINI has received the PhD degree in Computer Engineering from Sapienza, University of Rome in 2014. His research interests are on simulation on parallel and distributed architectures, a field in which he has more than 60 publications on books, journals and conference proceedings. In 2015, he has won the prize for the best PhD thesis of the year from Sapienza, University of Rome. He has worked as a researcher in many national/international research institutes, such as CINI, CINFAI, IRIANC, and BSC. His e-mail address is pellegrini@diag.uniroma1.it.

FRANCESCO QUAGLIA has received the PhD degree in Computer Engineering in 1999 from Sapienza University of Rome. He works as a Full Professor at the School of Engineering of University of Rome Tor Vergata. His research interests include distributed systems and protocols, middleware platforms, parallel/distributed and federated simulation systems, parallel computing, fault tolerance, and transactional systems. In these areas, he has been an author of more than 120 technical articles. He has been coordinator within several EU and international/national projects. His e-mail address is francesco.quaglia@uniroma2.it.