A Performance Evaluation of the Lightweight Time Warp Protocol in Optimistic Parallel Simulation of DEVS-based Environmental Models

Qi Liu and Gabriel Wainer
Department of Systems and Computer Engineering
Carleton University Centre on Visualization and Simulation (V-Sim)
Carleton University, Ottawa, Canada
{liuqi, gwainer}@sce.carleton.ca

Abstract

The Lightweight Time Warp (LTW) protocol offers a novel approach to high-performance optimistic parallel discrete-event simulation, especially when a large number of simultaneous events need to be executed at each virtual time. With LTW, the local simulation space on each node is partitioned into two sub-domains, allowing purely optimistic simulation to be driven by only a few full-fledged logical processes (LPs), while most processes are turned into lightweight LPs, free from the burden associated with Time Warp (TW) execution. This paper presents a comparative performance evaluation of the TW and LTW protocols for simulating several DEVS-based environmental models. The experiments indicate that the LTW protocol improves performance in terms of shortened execution time, reduced memory usage, lowered operational cost, and enhanced system stability.

1. Introduction

The Time Warp (TW) mechanism [1] has been widely used to speed up parallel discrete-event simulation (PDES) systems (e.g., [2-3]). A TW simulation is executed by several asynchronous logical processes (LPs). Each LP has its own Local Virtual Time (LVT) and interacts with other LPs via time-stamped event messages. Potential causality errors are detected and recovered by rollbacks that are triggered by straggler or anti-messages. To this end, each LP also maintains *persistent* input, output, and state queues to store historical events and states, which can only be discarded when the Global Virtual Time (GVT) has advanced beyond their time stamps. A survey of PDES techniques is provided in [4].

The increasing scale and complexity of PDES systems poses new demands on the TW mechanism. With many LPs loaded on each available node, saving historical data in the persistent queues not only consumes enormous memory, but also increases the cost of queue operation, fossil collection, and dynamic process migration. Further, the conventional rollback mechanism relies solely on propagation of anti-messages to cancel incorrect computation at the LPs, putting a tremendous burden on the communication infrastructure and, as the number of LPs involved in the rollback increases, impairing the performance and scalability of the entire system. To address these problems, the Lightweight Time Warp (LTW) protocol has been proposed in [5] to improve simulation performance in a variety of ways, including shortened execution time, reduced memory footprint, lowered operational cost, and enhanced system stability.

The Discrete Event System Specification (DEVS) [6] is a general methodology for describing discrete-event systems. P-DEVS [7] extends DEVS to handle simultaneous events, increasing the parallelism in the simulation. Cell-DEVS [8] defines n-dimensional cell spaces as discrete-event models. TW simulation of DEVS models is still uncommon in the literature [9-12]. The PCD++ environment [13] is one of the recent efforts to support TW simulation of DEVS and Cell-DEVS models based on the WARPED kernel [14]. In this research, we have extended PCD++ to include the LTW concepts. This paper compares the performance of LTW and standard TW protocols for simulating DEVS-based environmental systems in PCD++.

The rest of the paper is organized as follows. Section 2 introduces the motivation and related work. Section 3 recaps LTW protocol in the context of PCD++. Section 4 gives the performance analysis. Conclusion and future work are presented in Section 5.

2. Motivation and Related Work

Although a number of studies have been devoted to improving the efficiency of the TW protocol, several issues remain to be resolved.

1087-4097/09 $25.00 © 2009 IEEE
DOI 10.1109/PADS.2009.15

One issue is to maintain high-performance execution even when system memory is tight. Techniques such as *pruneback*, *cancelback*, and *artificial rollback* allow the system to recover from a memory stall [4], but only at the expense of increased computation and communication cost. Various fossil collection schemes have been used to reclaim past events and states (e.g., [15-16]). However, fossil collection still constitutes a major overhead simply because the number of LPs and fossil data increase dramatically in large-scale simulations. Further, it might be unable to reclaim fossil data before memory exhaustion if the GVT does not advance sufficiently fast, especially when a massive number of simultaneous events are executed at each virtual time.

The pitfalls of *rollback echoes*, *chasing hazards* and *cascading rollbacks* have been discussed in numerous studies [4]. Optimism control is one way to improve rollback efficiency (e.g., [17-18]), sacrificing the degree of parallelism to a certain extent. Rollback cost can also be reduced by different cancellation algorithms (e.g., [19-20]), which inevitably increase computational complexity of the system.

Various data structures have been investigated to facilitate event queue operations (e.g., [21-22]). While these approaches can certainly be used to improve performance, an attractive alternative solution is to keep the event queues short throughout the simulation.

Agile process migration is needed in parallel systems to reduce the communication cost and to minimize the interference with normal system execution. Even more so in large-scale TW systems where a potentially unbounded number of events and states associated with an LP must be transferred. An early work proposed in [23] employed a phase-based computation model to reduce LP migration cost. Yet this scheme suffers from increased message routing overhead as any LP can be divided into small chunks spread all over the system. More recently, an event reconstruction algorithm was presented in [24] so that only the state queue needs to be transferred. Nevertheless, this approach only works for systems with fine event granularity and small state size.

The way how simultaneous events are handled has serious implication on both simulation correctness and performance [25]. To ensure correct simulation results, P-DEVS introduces (partial) causal dependency among simultaneous events, requiring a control flow to enforce an orderly event execution at each virtual time. From a performance perspective, however, this expanded execution of simultaneous events could increase the overhead for state-saving, rollback, and fossil collection, an issue that has not yet been addressed by existing TW-based DEVS systems [9-12].

To address these challenges in a systematic way, the LTW protocol [5] takes a novel approach that isolates most LPs from the TW mechanism, while the purely optimistic simulation is driven by only a few full-fledged TW LPs. In this paper, we analyze the effectiveness of the LTW protocol quantitatively.

3. The Lightweight Time Warp Protocol

This section briefly recaps the LTW protocol in the context of PCD++, while detailed LTW algorithms with pseudo-code can be found in [5].

3.1. Concepts and Assumptions

P-DEVS supports hierarchical model construction in a modular way. A system is described as a hierarchy of behavioral (*atomic*) and structural (*coupled*) model components. The LPs are specialized into *simulators* and *coordinators*, forming a hierarchy that mimics the model structure [7]. A simulator executes the DEVS functions defined in its associated atomic model, while a coordinator is attached to a coupled model to control the simulation in line with the P-DEVS formalism.

PCD++ adopts a flat LP structure that creates a *Node Coordinator* (NC), a *Flat Coordinator* (FC), and a set of *Simulators* on each node [13]. Doing so eliminates intermediary coordinators in the LP hierarchy, with reduced communication cost. The NC is the local central controller and the endpoint of inter-node communication, whereas the FC routes messages between its child Simulators and the parent NC. The event-processing algorithms can be found in [13].

Six types of events are defined to execute the simulation in a message-driven fashion: External (*x, t*) and output (*y, t*) messages encode the actual model data, while initialization (*I, t*), collect (*@, t*), internal (**, t*), and done (*D, t*) messages control the execution of simultaneous events at each virtual time.

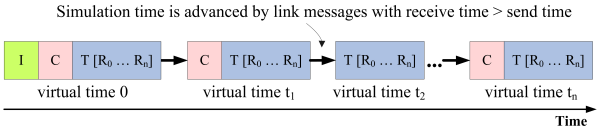

Figure 1. Simulation process on a node

Figure 1 gives a high-level overview of the simulation process on a node. At any virtual time, the message flow among the LPs is organized into a multi-phased structure that includes an optional *collect phase* (C) and a mandatory *transition phase* (T), which in turn may involve multiple rounds of computation ($[R_0 \ldots R_n]$) to execute state transitions incrementally. The LPs are initialized with an *initialization phase* (I) at virtual time 0. At the end of each transition phase,

the NC sends link messages to the local LPs, advancing the virtual time on the node. A closer look at the message-passing organization is presented in [5].

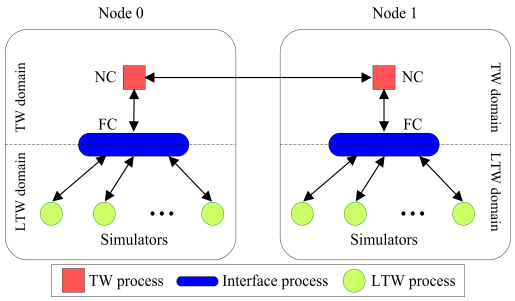

Figure 2. Division of domains in LTW

As illustrated in Figure 2, the novelty of the LTW protocol concerns the division of the local simulation space into two sub-domains, namely a *TW domain* and a *LTW domain*. The former contains full-fledged LPs that execute based on the standard TW protocol. In PCD++, the **NC** is the only full-fledged LP on each node. On the other hand, the latter contains lightweight LPs that are released from the burden of TW execution. The LTW domain is where all the PCD++ **Simulators** reside. LPs from different domains interact via a mixed-mode interface LP that serves as a gatekeeper for the lightweight LPs. The interface LP is realized by the **FC** in PCD++. Note that all the LPs still execute asynchronously and optimistically with potentially different LVT values. However, by virtue of the LTW protocol, the lightweight LPs no longer rely on the persistent queues to recover from causality errors, with much lower operational overhead.

The LTW protocol is based on the following assumptions regarding the control of the LPs [5].

1. All communication of the Simulators goes through the FC (i.e. no direct communication between the Simulators). Hence, the FC has the full knowledge of the timing of state changes at the Simulators.

2. The virtual time on each node is advanced only by the LPs in the TW domain. That is, the FC and Simulators do not advance their LVTs voluntarily, nor do they send messages across virtual time boundaries.

3. Rollbacks always propagate from the TW domain to the LTW domain via the FC. This is actually implied by the other two assumptions since any speculative computation on a node is always initiated by full-fledged LPs in the TW domain and then spread to the LTW domain through the FC. Hence, the FC knows when rollbacks will occur at the Simulators.

3.2. Rule-based dual-queue event scheduling

As discussed earlier, keeping past events in the persistent input queue is one of the major sources of operational overhead in the TW mechanism. To solve this problem, the LTW protocol introduces an extra *volatile input queue* that does not preserve processed events at all. Specifically, it is used to hold temporarily the *simultaneous* events exchanged between the FC and the Simulators at each virtual time. These volatile events are deleted immediately after execution, reducing memory usage for saving past input events. Moreover, the volatile input queue only contains simultaneous events, allowing for efficient queue operation in O(1) time. At any virtual time, events are inserted into the volatile queue as the simulation moves into a collect or transition phase, and removed as the execution proceeds. By the end of the phase, the volatile queue becomes empty, making the queue relatively short throughout the simulation.

Consequently, the persistent input queue only stores events sent between the NC and FC, making the queue far shorter than it would be in the TW protocol with more efficient queue operations. Since the simultaneous events executed by the FC and Simulators at any virtual time are either committed or cancelled together in the optimistic simulation, it is safe to exclude them from the persistent queue.

In addition, anti-messages need not to be saved for the volatile events, further reducing memory usage. Since potentially incorrect volatile events have already been deleted during forward execution, no event cancellations are required to roll back the Simulators. Hence, rollback propagation is effectively restricted to the TW domain only between the NC and FC on each node, with much lower overhead.

To schedule events in both queues properly, a scheduler is created on each node to determine the next event to be executed during each simulation cycle by evaluating the following rules.

Rule 1. Idle condition. The simulation becomes idle on the host node if the volatile queue is empty and the persistent queue does not contain events with time stamps before or at the simulation stop time. The simulation may be reactivated later upon the arrival of messages from the other nodes.

Rule 2. Simulation progress. The scheduler selects the next persistent event with a time stamp earlier than the simulation stop time if the volatile queue becomes empty so that the NC can 1) advance simulation time on the node, or 2) resume forward execution from the unprocessed persistent events after a rollback, or 3) reactivate the simulation from the idle state upon the arrival of remote messages, which are inserted into the persistent queue.

Rule 3. Aggressive inter-node communication. During a collect phase, the NC may send messages to remote nodes. As these are potentially straggler

messages at the receiving end, a delay in their delivery could postpone rollbacks at the destination, resulting in degraded performance. Thus, the scheduler grants a higher priority to the persistent events than those volatile events with the same time stamp in order to process inter-node messages immediately.

Rule 4. LTSF execution. The next volatile event is selected to execute in all other cases, enforcing a Least-Time-Stamp-First execution on the node.

3.3. Aggregated state management

LTW provides a new scheme that allows the Simulators to delegate the responsibility of state management to the FC. To this end, the FC uses an *aggregated state manager* that maintains not only the state queue for the FC itself, but also those used by the Simulators. In order to identify the active Simulators whose states have been modified at the current virtual time, a Simulator's state queue is associated with a *dirty bit*, which is set whenever the FC sends an event to the corresponding Simulator. The actual state saving is carried out only when the FC detects that the events previously sent to the Simulators have already been processed, and is performed only for those Simulators with dirty bits set to true. No dirty bit is used for the FC's own state queue because the FC is always involved in the computation at each virtual time. The dirty bit is reset after saving the state of a Simulator.

With this state management scheme, an optimal risk-free state-saving strategy was proposed so that only a single state is saved for, and only for, an active LP at each virtual time [5]. It is risk-free as no performance penalty is incurred as a result of saving fewer states. To implement this strategy in PCD++, a new *state-saving phase* is added after each transition phase. Before the NC advances simulation time, it first instructs the FC to save states for the current virtual time. As all simultaneous events at the current virtual time have been processed at this moment, the saved states contain the latest values of the state variables. Only after the state-saving phase, can the NC send link messages to the FC to advance the simulation time on the node. The NC saves its own state using whatever strategy implemented in the TW domain.

3.4. Lightweight rollback mechanism

In LTW, the Simulators are turned into truly lightweight LPs whose input events become volatile, whose output queues are removed altogether, and whose state queues are delegated to the FC. The only operation required for the Simulators to recover from causality errors is state restoration, which is performed by the FC on behalf of the Simulators.

To this end, the FC uses an array of *latest state change time* (LCT) to keep track of the latest times when state transitions are triggered by (*, t) messages at the Simulators. The LCT value is updated whenever the FC sends a (*, t) message to a Simulator. During rollbacks, the FC first cancels its speculative interactions with the NC based on standard TW. It then invokes the scheduler to delete all volatile events scheduled at or after the rollback time. Finally, the FC instructs its aggregated state manager to recover the state for each Simulator whose LCT is greater than or equal to the rollback time. After the state restoration, the LCT is reset to the LVT of the recovered state. Thus, rollbacks can be performed efficiently due to the elimination of secondary rollbacks at the Simulators.

3.5. LTW implications

Though largely a local control protocol, LTW also has an impact on several aspects of the TW global control mechanism. First, fossil collection on each node is accelerated, not only because the fossil data in the persistent queues are minimized, but also because most of the states are managed in a centralized manner, allowing for efficient batch operations. Secondly, agile process migration is possible since only the state queues need to be transferred to move the lightweight LPs around in dynamic load balancing. The appropriate decision points for process migration would be at the end of each state-saving phase when all the volatile events have been executed (and deleted) and the states of the LPs have been saved.

Additionally, LTW can be seamlessly integrated with other TW optimizations to further improve performance. For instance, various state-saving and cancellation strategies can be applied to the TW domain directly. In a way, LTW can be considered as complimentary to the Local Time Warp [17] in the sense that the former is a purely optimistic approach to reducing operational cost within each local simulation space, while the latter is a locally optimistic approach to mitigating cascaded rollbacks in the global space. It is easy to combine both approaches in a consistent way.

On the applicability of the LTW protocol, we stress that, though only a single LTW domain is considered in this paper, the protocol can be readily extended to support hybrid systems that require multiple LTW domains coexisting on each node to implement domain-specific formalisms. Besides, the basic concepts derived from the LTW protocol could also apply to a wide range of TW systems through carefully choosing the level of event granularity and imposing an appropriate control over the LPs.

4. Performance Evaluation

4.1. Experiment platform and metrics

Both TW and LTW protocols have been implemented in PCD++. A stress test was carried out on a cluster of 28 HP Proliant DL140 servers (dual 3.2GHz Intel Xeon processors, 1GB 266MHz RAM with 2GB disk swap space) running on Linux WS 2.4.21 and communicating over Gigabit Ethernet using MPICH 1.2.7. Note that severe memory swapping may occur if the memory usage approaches to the upper limit of 1GB on a node.

Table 1. Performance metrics

Metrics	Description
T	Total execution time of the simulation (sec)
MEM	Maximum memory consumption (MB)
PEE	Number of events executed in persistent queue
VEE	Number of events executed in volatile queue
PQLen	Average length of the persistent input queue
VQLen	Average length of the volatile input queue
SS	Total number of states saved
OPT-SK	Number of states reduced by the optimal strategy
FCT	Average time spent on a single fossil collection (ms)
PriRB	Number of primary rollbacks
SecRB	Number of secondary rollbacks
RB	Total number of rollbacks (i.e., PriRB + SecRB)
EI	Number of events imploded in persistent queue
ER	Number of events unprocessed in persistent queue

Table 1 lists the metrics collected in the experiments through extensive instrumentation and measurement. The experimental results for each test case were averaged over 10 independent runs to strike a balance between data reliability and testing effort[1]. For those test cases executed on multiple nodes, the results were also averaged over the participating nodes to obtain a *per-node* evaluation. The queue lengths (i.e., PQLen and VQLen) were averaged over samples collected every 20 event insertions in the queues.

4.2. Environmental models

Three Cell-DEVS models with varied characteristics were validated and tested. Two of them simulate stationary wildfire propagation over 50 hours in a 2D cell space based on the Rothermel model [26]. However, they differ in the way the spread rates are calculated. The first fire model, referred to as *Fire1*, uses predetermined rates at reduced runtime computation cost (see [27] for the model definition). The second fire model, referred to as *Fire2*, invokes the `fireLib` library [28] to calculate spread rates dynamically, with higher runtime computation density, based on a set of parameters such as fuel type, moisture, wind direction and speed. The time for executing a (**, t*) message at the Simulators, which reflects the computation intensity of the state transitions, was calibrated at 112 and 748 μs for *Fire1* and *Fire2* respectively.

The other model, called as *Watershed*, simulates environmental influence on hydrological dynamics of water accumulation over 30 minutes in a 3D cell space [27]. Though it is not as compute-intensive as *Fire2* (577 μs state transition time), a larger neighborhood of 10 cells on different layers of the cell space is defined with increased communication intensity.

Unlike cellular automata models, which evaluate all the cells synchronously at discrete time steps, these Cell-DEVS models define the cell spaces as discrete-event models where each cell is an independent DEVS atomic model executed by a Simulator in an event-driven fashion, allowing for efficient asynchronous execution without losing simulation accuracy.

As in many other DEVS systems, these Cell-DEVS models execute a great number of simultaneous events at each virtual time, increasing the operational cost of TW simulation considerably. In the next section, we will show that the LTW protocol is well-suited for improving simulation performance in such situations.

4.3. Test results and analysis

The comparative evaluation was conducted under the same configurations. Both protocols used aggressive cancellation, copy state-saving optimized with Message Type-based State Saving (MTSS) [13], and pGVT algorithm [14]. In addition, the optimal risk-free state-saving strategy introduced in Section 3 was enabled for the LTW protocol. In all test cases, message logging activities were turned off to minimize the impact of file I/O operations on system performance. Also, the corresponding test cases used the same partition scheme that divides the cell spaces into horizontal rectangles (or rows) as evenly as possible among the compute nodes.

In the following tables, a "×" mark indicates a failed test case due to memory exhaustion, while a shaded entry attributes the poor performance to severe memory swapping activities. A "–" mark stands for a case that was not tested because either the performance trend is already clear in the series, or the model cannot be divided further with the given partition scheme. The best execution time is highlighted in each series. The results (T and MEM) of a sequential simulator are also provided as a reference for evaluating the absolute performance of both protocols.

[1] An expanded experiment is being conducted to attain a 95% confidence interval for all the test cases.

Table 2. Total execution time and maximum memory usage for Fire1

Size	Sequential	Prot.	Metric	1	2	4	6	8	10	12	14	16	18	20	22	24	26	28
50×50	5.54 (T) 29.11 (MEM)	TW	T	×	9.08	5.87	5.26	**5.01**	5.39	5.49	5.55	5.95	–	–	–	–	–	–
			MEM	×	813.57	220.42	109.94	61.79	43.73	34.84	26.37	22.22	–	–	–	–	–	–
		LTW	T	5.78	3.61	3.02	2.98	**2.78**	3.01	3.23	3.25	3.54	–	–	–	–	–	–
			MEM	63.53	65.83	27.42	20.58	14.25	13.24	11.98	9.95	9.31	–	–	–	–	–	–
100×100	56.07 (T) 110.59 (MEM)	TW	T	×	×	2749.13	484.91	40.09	35.66	34.46	**32.35**	33.51	32.53	32.44	33.4	35.0	35.19	35.96
			MEM	×	×	2279.42	1492.31	882.82	576.61	410.19	307.79	244.6	197.97	162.92	137.77	121.47	103.03	91.75
		LTW	T	78.21	43.84	31.62	24.35	23.58	22.61	22.26	**21.62**	21.86	21.88	22.03	22.2	22.0	22.46	21.76
			MEM	405.5	373.25	271.62	160.26	110.94	82.65	66.75	55.65	48.18	43.55	38.92	36.22	34.05	32.3	29.94
150×150	260.65 (T) 242.69 (MEM)	TW	T	×	×	×	×	×	1516.48	893.43	572.83	314.03	202.71	141.46	**140.98**	142.63	142.01	143.18
			MEM	×	×	×	×	×	2309.12	1935.02	1449.83	1131.65	906.07	744.91	623.9	527.05	460.76	404.44
		LTW	T	1489.77	517.92	394.56	122.44	112.93	110.63	111.7	109.67	107.02	107.23	105.27	107.1	106.75	104.88	**104.74**
			MEM	1418.85	1294.08	986.62	660.31	415.01	296.96	230.4	186.68	161.7	137.22	123.85	105.07	96.8	90.88	85.09
200×200	815.43 (T) 432.13 (MEM)	TW	T	×	×	×	×	×	×	×	×	×	×	4324.31	1236.26	1065.79	881.61	**737.14**
			MEM	×	×	×	×	×	×	×	×	×	×	1848.93	1560.7	1528.73	1188.06	1058.7
		LTW	T	12571.7	6894.36	1425.16	920.86	646.56	350.58	334.77	331.2	333.12	326.7	327.56	327.46	**322.93**	330.03	327.24
			MEM	1679.36	1644.54	1393.66	1229.82	1145.6	805.17	582.49	431.18	393.15	291.49	244.01	209.52	235.47	186.1	188.68

Table 2 gives the resulting total execution time and maximum memory usage (T and MEM) for *Fire1* of varied sizes on different number of nodes. It is clearly shown that the LTW protocol outperforms its TW counterpart in all successful cases. First, the maximum memory usage on each node is reduced by 45% up to 92%, making it possible to execute the model using a smaller number of nodes, with significantly lower simulation cost. Secondly, the total execution time is decreased by 24% up to 60% among those test cases with sufficient memory, and this outstanding improvement in execution time is achieved with a much smaller memory footprint at the same time.

To find out the reason that causes the differences, the other metric values are compared. As an example, we present a comparison of the 100×100 *Fire1* on 14 nodes using the collected metrics, shown in Table 3.

Table 3. 100×100 Fire1 on 14 nodes

Metrics	TW	LTW	LTW vs. TW
PEE	96685.07	10597.71	
VEE	0	67214.07	
PQLen	24798.12	2636.95	↓ **89.37%**
VQLen	0	121.89	
SS	52819.64	22675.14	↓ **57.07%**
OPT-SK	0	18445.36	
FCT	488.14	84.15	↓ **82.76%**
PriRB	613.14	604.00	↓ **1.49%**
SecRB	11922.07	981.14	↓ **91.77%**
RB	12535.21	1585.14	↓ **87.35%**
EI	61751.93	5826.36	↓ **90.56%**
ER	48118.79	5790.93	↓ **87.97%**

Thanks to the introduction of the volatile input queue, the average length of the persistent input queue is shortened significantly by 89.37%, reducing the overhead of queue operations and memory consumption considerably. On the other hand, the volatile queue is kept short throughout the simulation with an average length of just 121.89 events, despite the fact that a majority of 86.38% input events executed on each node have been turned into volatile under the LTW protocol.

Owning to the optimal risk-free state-saving strategy, which reduces the number of state-saving by 44.86% on top of the MTSS strategy, the total number of states saved in the LTW case is 57.07% fewer than in the TW case, resulting in less memory usage as well.

As expected, the time for each fossil collection is decreased from 488.14 ms to just 84.15 ms, a dramatic reduction of 82.76%.

When comparing the rollback performance, the LTW protocol again shows a big advantage over the TW counterpart. The number of secondary rollbacks is reduced by 91.77%, showing that rollback propagation is effectively contained within the TW domain on each node. Moreover, the number of primary rollbacks is reduced slightly by 1.49%, which, combined with the fact that the total number of events executed on each node (i.e., PPE + VEE) is decreased by 19.52%, suggests a more stable system with less speculative computation. Consequently, the numbers of events imploded and unprocessed in the persistent queue are also declined by around 90%, further accelerating the rollback operations.

The experimental results for the *Fire2* and *Watershed* models are shown in Table 4 and Table 5 respectively. Again, LTW reduces maximum memory consumption by approximately 34% up to 92% for *Fire2* and by 73% up to 93% for *Watershed*. The reduction in memory usage is more prominent for *Watershed* largely because, with a higher number of simultaneous events exchanged between the LPs at each virtual time, a larger percentage of states are reduced with the optimal state-saving strategy.

Table 4. Total execution time and maximum memory usage for Fire2

Size	Sequential	Prot.	Metric	1	2	4	6	8	10	12	14	16	18	20	22	24	26	28
50×50	19.29 (T) 29.52 (MEM)	TW	T	×	20.89	13.93	12.19	10.91	**10.41**	10.8	10.64	10.84	10.55	11.31	12.51	12.76	13.39	13.44
			MEM	×	800.26	226.82	108.41	65.37	46.29	34.54	28.13	23.23	20.19	18.19	16.31	14.81	13.72	12.85
		LTW	T	20.26	14.23	10.38	9.69	9.46	8.84	9.01	8.51	8.64	8.4	**8.32**	9.28	9.34	9.51	10.27
			MEM	81.24	66.92	34.99	22.6	17.77	14.65	13.02	11.76	10.83	10.17	9.63	9.29	8.99	8.73	8.49
100×100	119.95 (T) 109.57 (MEM)	TW	T	×	×	3284.37	460.32	68.67	54.63	52.03	48.92	48.58	46.96	**46.37**	47.53	48.69	49.39	49.97
			MEM	×	×	2159.1	1319.08	658.14	576.72	411.14	310.95	240.42	198.4	163.47	149.65	112.23	99.94	83.78
		LTW	T	206.16	114.98	60.09	54.37	51.22	44.11	41.61	40.37	38.87	37.55	**35.54**	36.83	36.23	36.46	36.48
			MEM	314.37	285.18	248.32	137.73	102.24	81.63	65.57	54.35	48.91	45.62	42.6	38.42	35.75	33.84	32.03
150×150	414.25 (T) 243.71 (MEM)	TW	T	×	×	×	×	×	4448.08	2487.95	651.06	394.92	244.97	167.25	**164.79**	167.42	165.64	168.88
			MEM	×	×	×	×	×	1817.71	1375.23	1399.3	1086.72	905.96	744.91	562.55	532.14	425.91	399.51
		LTW	T	1592.43	493.61	223.65	178.2	174.63	165.84	168.66	167.14	140.67	140.21	137.0	134.3	136.11	**133.1**	134.01
			MEM	1210.37	924.16	641.79	586.92	385.41	269.62	205.4	172.18	139.44	122.16	112.93	104.22	94.47	89.39	85.79
200×200	1033.61 (T) 424.96 (MEM)	TW	T	×	×	×	×	×	×	×	×	×	12112.7	3206.02	1501.28	1202.48	900.05	**764.21**
			MEM	×	×	×	×	×	×	×	×	×	1943.55	1785.9	1618.94	1522.69	1475.58	1243.95
		LTW	T	11707.5	3363.07	1339.92	1173.69	562.68	414.52	412.92	412.89	381.1	376.58	417.44	373.11	372.6	**370.04**	371.56
			MEM	1661.95	1562.62	1267.71	1292.97	885.61	438.81	363.5	313.96	289.68	274.55	240.98	227.23	208.62	192.61	173.08

Table 5. Total execution time and maximum memory usage for Watershed

Size	Sequential	Prot.	Metric	1	2	4	6	8	10	12	14	16	18	20	22	24	26	28
15×15×2	258.27 (T) 43.99(MEM)	TW	T	×	×	2059.62	899.49	**84.97**	87.06	86.59	88.76	–	–	–	–	–	–	–
			MEM	×	×	1718.02	997.21	691.2	536.37	422.49	333.53	–	–	–	–	–	–	–
		LTW	T	262.99	171.18	112.69	100.54	**79.45**	82.27	82.08	82.59	–	–	–	–	–	–	–
			MEM	45.66	27.91	148.48	121.54	128.96	113.14	101.29	90.39	–	–	–	–	–	–	–
20×20×2	471.86 (T) 72.67 (MEM)	TW	T	×	×	×	×	2451.7	857.3	757.65	724.55	**638.97**	676.42	–	–	–	–	–
			MEM	×	×	×	×	1618.94	1180.67	967.51	778.53	643.52	535.21	–	–	–	–	–
		LTW	T	473.81	268.87	181.94	155.09	140.14	**104.77**	108.52	109.58	110.35	112.87	–	–	–	–	–
			MEM	76.02	40.04	164.35	136.36	130.82	149.81	137.24	129.85	115.87	111.99	–	–	–	–	–
25×25×2	735.39 (T) 115.48 (MEM)	TW	T	×	×	×	×	×	×	×	2002.73	1948.95	1922.21	1705.19	1597.08	**1585.6**	–	–
			MEM	×	×	×	×	×	×	×	1519.54	1434.77	1262.59	1063.03	774.38	663.21	–	–
		LTW	T	748.49	469.65	306.25	257.18	195.16	176.19	172.39	**136.18**	136.37	142.69	143.86	139.54	141.85	–	–
			MEM	119.8	70.46	164.86	128.68	131.07	132.81	132.27	153.82	141.87	128.25	114.39	113.95	103.44	–	–
30×30×2	1041.39 (T) 168.46 (MEM)	TW	T	×	×	×	×	×	×	×	×	5381.55	4475.37	3133.72	3130.89	2920.06	**2765.2**	2784.83
			MEM	×	×	×	×	×	×	×	×	2192.96	1867.83	1602.25	1388.87	1206.87	1055.31	924.49
		LTW	T	1098.11	616.28	390.68	293.33	237.82	208.26	204.82	198.27	169.12	168.45	168.01	165.54	165.64	166.55	**162.43**
			MEM	174.08	89.69	163.07	164.18	151.55	171.62	148.91	138.31	117.57	139.45	156.5	149.91	130.2	122.69	114.69

Table 6. 100×100 Fire2 on 20 nodes

Metrics	TW	LTW	LTW vs. TW
PEE	68346.55	11658.75	
VEE	0	56057.00	
PQLen	17533.37	2149.91	↓ **87.74%**
VQLen	0	75.31	
SS	33833.00	17565.40	↓ **48.08%**
OPT-SK	0	15591.10	
FCT	245.12	58.36	↓ **76.19%**
PriRB	769.95	740.55	↓ **3.82%**
SecRB	12794.35	2036.45	↓ **84.08%**
RB	13564.30	2777.00	↓ **79.53%**
EI	46877.55	7197.90	↓ **84.65%**
ER	29512.45	6651.60	↓ **77.46%**

Table 7. 20×20×2 Watershed on 18 nodes

Metrics	TW	LTW	LTW vs. TW
PEE	1253641.94	361457.78	
VEE	0	856256.00	
PQLen	334016.67	77790.62	↓ **76.71%**
VQLen	0	26.04	
SS	371273.33	73186.94	↓ **80.29%**
OPT-SK	0	288247.50	
FCT	61313.67	395.63	↓ **99.35%**
PriRB	173.50	159.94	↓ **7.81%**
SecRB	22816.67	2165.33	↓ **90.51%**
RB	22990.17	2325.28	↓ **89.89%**
EI	625210.33	175521.11	↓ **71.93%**
ER	569337.94	172280.33	↓ **69.74%**

For those cases with sufficient memory, the total execution time is decreased by 13% up to 32% for *Fire2* and by 5% up to 91% for *Watershed*. A general trend reflected in the experimental results is that the reduction in execution time and memory usage is greater for models with larger sizes, indicating an improved scalability.

The other metric values for the 100×100 *Fire2* on 20 nodes and 20×20×2 *Watershed* on 18 nodes are given in Table 6 and Table 7 respectively. As we can see, a similar pattern can be observed regarding the improvement of the metrics, suggesting that the LTW protocol is suitable for simulating models with varied computation and communication characteristics.

In terms of absolute performance, the LTW cases attain higher and more consistent speedup than the TW cases. In some scenarios, the performance of a TW simulation is even worse than the sequential execution (e.g., 50×50 *Fire1* on 2 and 4 nodes; 20×20×2 *Watershed* on 14, 16, and 18 nodes) mainly due to the excessive communication and operational overhead. However, such scenarios do not arise in

the LTW cases tested in our experiment.

5. Conclusion and future work

The LTW protocol offers a novel approach that systematically addresses several important issues of TW-based optimistic PDES systems, especially for DEVS-based simulations that require a large number of simultaneous events to be executed at each virtual time. It allows purely optimistic simulation to be driven by only a few full-fledged TW LPs, preserving the dynamics of the TW mechanism, while at the same time, accelerating the execution in each local simulation space significantly.

This paper presented an extended version of the PCD++ environment based on the LTW protocol. A comparative performance analysis has been conducted to evaluate both TW and LTW protocols in simulating several DEVS-based environmental models with different characteristics. The experimental results demonstrated that the LTW protocol outperforms the TW counterpart in various aspects, including shortened execution time, reduced memory usage, lowered operational cost, and enhanced system stability and scalability. We are currently working on integrating the LTW protocol with other TW optimizations to further improve performance. By taking advantages of the LTW protocol, we are also investigating dynamic process creation, deletion, and migration schemes to support more efficient load balancing as well as runtime structure changes in optimistic DEVS systems.

6. References

[1] D. R. Jefferson, "Virtual time". ACM Transactions on Programming Languages and Systems (TOPLAS) 7(3), 1985, pp. 404-425.

[2] J. S. Steinman, "The WarpIV simulation kernel". Proceedings of PADS, 2005, pp. 161-170.

[3] K. S. Perumalla, "μsik – A micro-kernel for parallel/distributed simulation systems". Proceedings of PADS, 2005, pp. 59-68.

[4] R. M. Fujimoto, "Parallel and Distributed Simulation Systems". Wiley-Interscience, 2000.

[5] Q. Liu, G. Wainer, "Lightweight Time Warp – A Novel Protocol for Parallel Optimistic Simulation of Large-Scale DEVS and Cell-DEVS Models". Proceedings of DS-RT, 2008, pp. 131-138.

[6] B. P. Zeigler, et al., "Theory of Modeling and Simulation". Academic Press, London, 2000.

[7] A. C. Chow, B. P. Zeigler, "Parallel DEVS: A parallel, hierarchical, modular, modeling formalism". Proceedings of WSC, 1994, pp. 716-722.

[8] G. Wainer, N. Giambiasi, "N-dimensional Cell-DEVS models". Discrete Event Dynamic Systems 12(2), 2002, pp. 135-157.

[9] B. P. Zeigler, et al., "Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions". In Fall Simulation Interoperability Workshop, SIW, 1999.

[10] J. Nutaro, "Risk-free optimistic simulation of DEVS models". Military, Government, and Aerospace Simulation Symposium, ASTC, 2004, pp. 113-118.

[11] K. H. Kim, W. S. Kang, "CORBA-based, multi-threaded distributed simulation of hierarchical DEVS models: Transforming model structure into a non-hierarchical one". ICCSA, LNCS 3046, 2004, pp. 167-176.

[12] Y. Sun, J. Nutaro, "Performance improvement using parallel simulation protocol and Time Warp for DEVS based applications". Proceedings of DS-RT, 2008, pp. 277-284.

[13] Q. Liu, G. Wainer, "Parallel environment for DEVS and Cell-DEVS models". SIMULATION 83(6), 2007, pp.449-471.

[14] R. Radhakrishnan, et al., "An object-oriented Time Warp simulation kernel". ISCOPE, LNCS 1505, 1998, pp. 13-23.

[15] V. Y. Vee, W. J. Hsu, "Pal: A new fossil collector for Time Warp". Proceedings of PADS, 2002, pp. 35-42.

[16] M. Chetlur, P. A. Wilsey, "Causality information and fossil collection in Time Warp simulations". Proceedings of WSC, 2006, pp. 987-994.

[17] H. Rajaei, "Local Time Warp: An implementation and performance analysis". Proceedings of PADS, 2007, pp. 163-170.

[18] M. Lees, et al., "Analysing probabilistically constrained optimism". Proceedings of DS-RT, 2006, pp. 201-208.

[19] Y. Zeng, et al., "Batch based cancellation: A rollback optimal cancellation scheme in Time Warp simulations". Proceedings of PADS, 2004, pp. 78-86.

[20] H. M. Soliman Ramadan, "Throttled lazy cancellation in Time Warp parallel simulation". SIMULATION 84(2-3), 2008, pp.149-160.

[21] J. Dahl, et al., "Event list management in distributed simulation". Euro-Par 2001 Parallel Processing, LNCS 2150, 2001, pp. 466-475.

[22] W. T. Tang, et al., "Ladder queue: An O(1) priority queue structure for large-scale discrete event simulation". ACM TOMACS, 2005, pp. 175-204.

[23] P. L. Reiher, D. R. Jefferson, "Virtual time based dynamic load management in the Time Warp operating system". SCS Multiconference on Distributed Simulation, 1990, pp. 103-111.

[24] L. Li, C. Tropper, "Event reconstruction in Time Warp". Proceedings of PADS, 2004, pp. 37-44.

[25] F. Wieland, "The threshold of event simultaneity". Proceedings of PADS, 1997, pp. 56-59.

[26] R. C. Rothermel, "A mathematical model for predicting fire spread in wild-land fuels". USDA Forest Service Research Paper, 1972, INT-115.

[27] G. Wainer, "Applying Cell-DEVS methodology for modeling the environment". SIMULATION 82(10), 2006, pp.635-660.

[28] C. D. Bevins, "fireLib User Manual and Technical Reference". http://www.fire.org/, accessed in Dec. 2008.