IMPROVING THE PERFORMANCE OF OPTIMISTIC TIME MANAGEMENT MECHANISM WITH SUB-STATE SAVING

B. Kaan Görür
Department of Computer Engineering,
Hacettepe University
Ankara, Turkey
bkaangorur@gmail.com

Kayhan İmre
Department of Computer Engineering,
Hacettepe University
Ankara, Turkey
ki@hacettepe.edu.tr

Halit Oğuztüzün
Department of Computer Engineering,
Middle East Technical University
Ankara, Turkey
oguztuzn@ceng.metu.edu.tr

Levent Yilmaz
Department of Computer Science and Software Engineering, Auburn University
Auburn, Alabama 36849, USA
yilmaz@auburn.edu

ABSTRACT

Optimistic approaches are scalable methods for time management in parallel and distributed simulations. In optimistic time management, logical processes advance their local time without constrained by the others in the simulation. If a logical process receives a message from its past, it goes back to a previously saved state, which is called a rollback. Even though the received message from the past would not cause any problem, rollback is performed in any case. In this paper, we presented a method to reduce the number of rollbacks, without sacrificing the accuracy of simulation. We propose to save a relatively small subset of the full simulation state to allow the logical processes to make a decision whether a rollback is really needed or not. Our technique is demonstrated in an agent-based simulation using the Time Warp algorithm adapted for optimistic time management for Repast HPC.

Keywords: parallel and distributed simulation, optimistic time management, Time Warp algorithm, state saving.

1 INTRODUCTION

Parallel and distributed processing techniques are used extensively to simulate models that require high computational power. In the last three decades, a lot of algorithms have been proposed to overcome performance issues in parallel and distributed simulations (PADS), for example Jefferson (1985), Leong, Agrawal, Agre (1993), Fleischmann and Wilsey (1995) and Collier and North (2012). Many studies show that improvements in the time management mechanism can bring about remarkable performance gains, see, for example Jafer and Wainer (2010) and Barnes et al. (2013). Two main approaches are popular in PADS community: conservative and optimistic time management. Conservative time management makes sure to keep logical processes (LP) safe against timing errors. LPs in conservative approach should advance their local time together, because they have to be in a correctly computed state. Therefore, once their local time is advanced, LPs are guaranteed not to receive any message from their past. Although the tight synchronization mechanism makes conservative time management more cautious, it causes the simulation

SpringSim-HPC 2017, April 23-26, Virginia Beach, VA, USA

to slow down in some cases. Therefore, several algorithms that loosen the synchronization have been offered. Optimistic time management is a strategy that does not strive for tight synchronization as much as conservative time management. Optimistic LPs are prone to receive messages from their past, known as straggler messages. Ignoring straggler messages may cause the simulation results to be incorrect, so LPs have to perform a rollback and take care of repairing their state when a straggler message is received. The main advantage of optimistic approaches is avoiding the overhead of synchronization operations. On the other hand, possibility of frequent rollbacks can be evaluated as the weakest point of optimistic approaches, because the cost of a rollback is considerable. Therefore, the performance of optimistic algorithms that have not been designed with special care to minimize the cost of rollbacks or that do not fit the application at hand may perform worse than conservative ones (Fujimoto 1999).

In optimistic algorithms, one of the two basic rollback methods can be employed to manage straggler messages. The first one is the incremental state saving method that makes LPs save state transitions. In this method, the LP that receives a straggler message makes reverse computations of the saved operations. The second method is copy-state saving that makes LPs to save their exact state. When a straggler message is received, the LP goes back to the saved state. In copy-state saving method, one of the difficulties is deciding on when checkpoints are taken. Taking a checkpoint at each time step is usually prohibitive in terms of both memory/disk and processors, although it provides LPs to directly restore their state to a past state. We will refer to this method as perpetual checkpointing, in the rest of this paper. To overcome this overhead, checkpoints are taken only at specified time ticks, known as periodic checkpointing. In this case, the LP restores the simulation state to the last checkpoint when a straggler message is received. Then, the events between the checkpoint and straggler's time are re-executed. This step is called coasting-forward. The coasting-forward step is the same as the normal execution of the simulation except that the send/receive operations are not repeated. After coasting forward is completed, the straggler message is processed and rest of the events are executed in order. Many studies showed that periodic checkpointing techniques can provide remarkable performance improvements (Lin et al. 1993; Fleischmann and Wilsey 1995). However LPs have to go back to the checkpoint that has been taken much earlier than time of the straggler message, in some cases. This means that coasting-forward step can be very expensive (Fujimoto 1999). In this paper, we are interested in only copy-state saving methods.

Both periodic and perpetual checkpointing have some pros and cons. We aim to leverage the advantages of those two approaches. In this paper, we propose a sub-state saving method to improve the performance of optimistic approach by reducing the number of rollbacks. The main contribution of this study is saving partial state of LPs to decide whether a rollback is really needed or not. As far as we know, no study in the literature uses state saving for this purpose. Instead, the other studies save states to create a checkpoint in case a straggler message is received in the future and LPs have to be restored (Fujimoto 1999; Fleischmann and Wilsey 1995). We also present an implementation strategy for this method on spatial agent-based models. The proposed method has been compared with conventional Time Warp algorithm for an agent-based model in a high performance computing (HPC) system. We experimented the proposed method in Repast HPC with Time Warp (RHPC_TW) (Görür et al. 2016) that is the earlier part of the present study.

2 RELATED WORK

Time Warp algorithm was proposed by Jefferson (1985) and became the most well-known optimistic time management algorithm. Performance evaluation studies for the Time Warp algorithm can be found in the literature. For example, Presley, Reiher and Bellenot (1990) observed a speedup of 29.5% for Sharks World with Time Warp. Perumalla (2007) and Bauer, Carothers and Holder (2009) evaluated the performance of Time Warp on more modern HPC systems. Barnes et al. (2013) broke simulation speed record in 2013 by employing Time Warp on the IBM's Sequoia supercomputer. Our previous study compared conservative and optimistic time management for agent-based simulations (Görür et al. 2016). In that study we showed that more scalable and faster results can be obtained by employing Time Warp.

Although Time Warp can reduce the execution time of many simulations, it is still open to improvements. Some performance optimizations were achieved by using adaptive checkpointing techniques. The goal of those techniques is finding the best checkpoint interval and dealing with the trade-off between the costs of state saving and coasting forward steps (Lin et al. 1993; Palaniswamy and Wilsey 1993; Rönngren and Ayani 1994; Fleischmann and Wilsey 1995). There are also some studies that relax causal constraints of LPs by ignoring some of the straggler messages and allowing incorrect computations (Thondugulam 1999). The challenge that arises from those studies is the trade-off between accuracy and speed. Another line of approach is examining the semantics of straggler messages. Leong, Agrawal and Agre (1993) considered the messages that can be run in any order. There are also approaches that employ Time Warp algorithm for various reasons, such as balancing the computational load among LPs (Glazer and Tropper 1993).

Spatial models are widely used in many agent-based simulations, such as the study of Epstein (2002). In that study, rebellion against authority has been modeled on a 2 dimensional grid. The usage of spatial-agent based models for socio-ecological systems was discussed by Filatova et al. (2013). Parallelization strategies for agent based models are examined by Fachada et al. (2016).

From the viewpoint of rollbacks and checkpointing, there are some overlaps between Time Warp and algorithm-based fault tolerant protocols. The main difference between them is that the main concern of algorithm-based fault tolerant protocols is detecting faults in processors and recover the system to improve the reliability of a multi-processor system (Yajnik and Jha 1997); while Time Warp deals with loosening the synchronization among LPs to improve the performance of a parallel or distributed simulation (Fujimoto 1999). On the other hand, a fault tolerant protocol for distributed systems that employs Time Warp was studied by Agrawal and Agre (1992).

In this study, we aim to improve the performance of Time Warp by not performing unnecessary rollbacks that are triggered as soon as a straggler message is received. The LPs that receive a straggler message has to start a rollback operation in conventional Time Warp algorithm to fix incorrect computations. Although some of the straggler messages do not contain any real interaction among agents, LPs perform a rollback in vain. Our method prescribes how to discover and ignore those messages to reduce the number of rollbacks without disturbing the accuracy of the simulation. Differently from the studies cited above, we saved partial states of the simulation to detect whether a rollback is needed or not.

3 IMPROVING THE PERFORMANCE OF TIME WARP

Figure 1 shows how the conventional Time Warp algorithm with periodic checkpointing works. In this example, two LPs, LP_A and LP_B, run in parallel. LP_B schedules an event to LP_A at time 13, but LP_A is notified at 20 about that event. By that time, LP_A has already processed the events between 13 and 20. LP_A is supposed to handle the event at 14 that was scheduled by LP_B, so LP_A should rollback. In the first step, LP_A restores its local state to the last saved state, in other words, the last checkpoint, and the local time of LP_A is set to the last checkpoint's time, which is 10 in this example. In the second step, the coasting-forward step is performed and the events between the checkpoint and the straggler message are processed. During the coasting forward, no communication operations are performed, because the messages have been already sent/received to/from other LPs. In the third step, the straggler message is processed in the correct order. In the fourth and last step, the events after the straggler message are re-executed and LP_A continues to its usual execution from time 14. Since this step is the usual execution of LPs, communication operations are allowed again. Before the rollback, LP_A might have sent some incorrect messages to some other LPs. During the fourth step, LP_A sends a special message, known as an anti-message, to these LPs so that they can undo the effect of the erroneous messages.

The total cost of a rollback in the conventional Time Warp algorithm is given in (1), where C_{SR}, C_{CF}, C_{RE}, and C_{AM} refer to the costs of state restoration, coasting forward, re-execution and anti-message handling steps, respectively.

$$TotalCost = C_{SR} + C_{CF} + C_{RE} + C_{AM} \quad (1)$$

To improve the performance of Time Warp, we propose extending it with saving sub-states in order to reduce the number of rollbacks. Before we introduce this method, as a preliminary, we discuss another one which we call Time Warp with state difference. Time Warp with state difference method inspects the straggler message and decides whether that message would affect the simulation or not. However, this method has a considerable overhead and does not bring a significant performance improvement. In the rest of this paper, we will refer to Time Warp with state difference method as state difference; and Time Warp with sub-state saving method as sub-state saving.

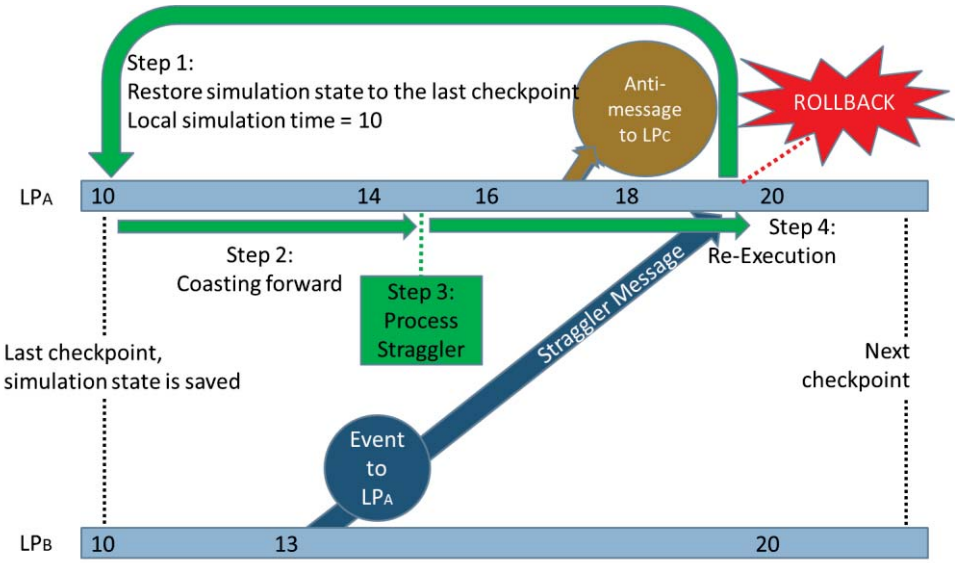

Figure 1: Flow of the Time Warp algorithm.

3.1 Time Warp with State Difference

An example flow of state difference extension of Time Warp is given in Figure 2, step by step. In this method, the goal is detecting if the straggler message would influence the simulation or not. When a straggler message is received, the LP saves its current state (step 1), in case the LP decides to ignore the straggler messages after the state difference. Then, state restoration and coasting forward steps are performed just like conventional Time Warp (steps 2 and 3). Differently from conventional Time Warp algorithm, the LP compares the "computed state" and the "must state" in the step 4. The computed state is the state that has already been computed before the straggler message is received; while the must state is the correct simulation state if the straggler message would have been processed at the right time. In order to access those states, the LP has to perform steps 2 and 3, because the state at straggler message's time had not been saved. Depending on the difference between the computed state and the must state, the LP decides whether a rollback is necessary or not. If a rollback is needed, then the usual flow of Time Warp is performed (steps 5 and 6). Otherwise, the LP ignores the straggler message and takes up where it left off before the straggler (alternative step 5 in Figure 2). That is why the LP saved its current state in step 1. Although the extra state saving (step 1) and state restoring (alternative step 5) operations are performed only once, they are still expensive, because all the agents in the LP have to be handled.

The main drawback of this method is extra state saving (step 1) and restoring operations (alternative step 4) in addition to restoration to the checkpoint and coasting forward which are still expensive. In other words, if a rollback is not necessary, step 2 and 3 are processed in vain. If the LP decides to avoid rollback, a state restoration will be required. The total cost of a rollback in the state difference method is given in (2), while C_{SS} and C_{SD} refer to the costs of state saving and state difference calculation steps, respectively.

$$TotalCost = C_{SS} + C_{SR} + C_{CF} + C_{SD} + \begin{cases} C_{RE} + C_{AM}, & if\ rollback\ is\ necessary \\ C_{SR}, & if\ rollback\ is\ not\ necessary \end{cases} \quad (2)$$

A rough comparison of the total costs of the conventional Time Warp algorithm and the state difference algorithm is as follows:

- If a rollback is needed, state difference method is more expensive than conventional Time Warp with the difference $C_{SS} + C_{SD}$.
- If a rollback is not needed, $C_{RE} + C_{AM}$ (in conventional Time Warp) and $C_{SS} + C_{SD} + C_{SR}$ (in state difference) should be compared. Since state saving and restoration operations are very expensive, state difference method will be worse than conventional Time Warp unless re-execution step is really long and expensive. Furthermore, to make state difference method perform better than conventional Time Warp, the difference between the must and computed states should be calculated by a light-weight method.

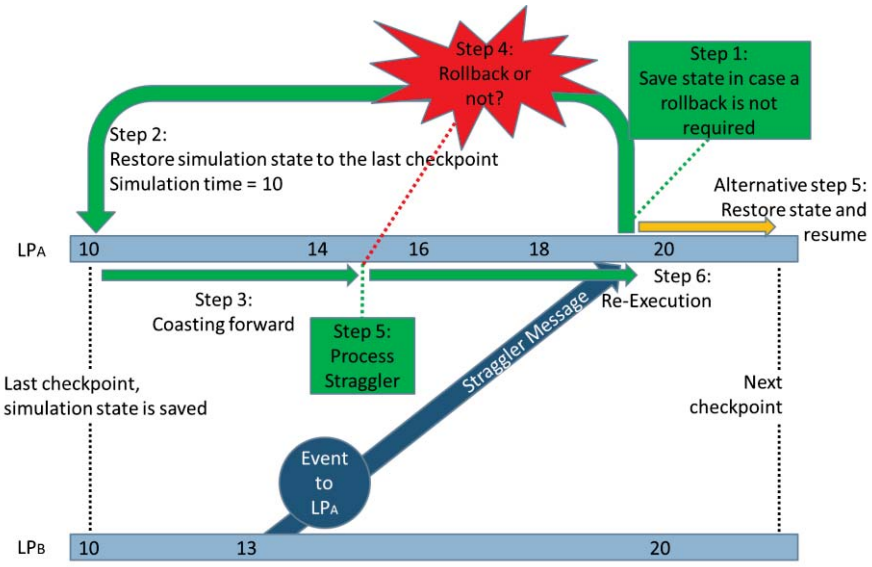

Figure 2: State difference algorithm.

In brief, the state difference extension may not worth in many cases; this is what led us into looking for a better alternative, the sub-state saving method.

3.2 Time Warp with Sub-state Saving

To overcome the drawbacks of the state difference method, straggler messages should be examined as they are received, instead of automatically going back to a past state. However, it is not always possible, because the LP does not know the earlier state of the simulation when it should have received the message. Therefore, the LP cannot find out whether the straggler message would contribute to simulation or not. To this end, we propose extending the Time Warp algorithm by sub-state saving method. Sub-state saving method prescribes all LPs to save a subset of their simulation state at every time step. These subsets are used to determine if a rollback is needed or not, when a straggler message is received. Therefore, sub-state saving method prevents an LP to start an immediate rollback. Instead, it postpones the rollback until straggler message inspection is completed. An example flow of this method is given in Figure 3.

Different from previous methods, the straggler message is inspected in step 1, as we see in Figure 1. At the end of step 1, LP_A makes a decision on performing a rollback or not. Thanks to sub-state saving method, LPs eliminate unnecessary rollbacks. The overhead of this method is saving sub-states at every time tick. Since sub-states are much smaller than full state of the simulation, the size of them is not so large. Therefore, saving sub-states is not as expensive as saving full states in terms of processors and disk. A detailed comparison between the sizes of sub-states and full states is given in the next section with an example. The

total cost of a rollback in sub-state saving method is given in (3), while C_{SMI} refers to the cost of straggler message inspection in step 1.

$$TotalCost = C_{SMI} + \begin{cases} C_{SR} + C_{CF} + C_{RE} + C_{AM}, & if\ rollback\ is\ necessary \\ 0, & if\ rollback\ is\ not\ necessary \end{cases} \quad (3)$$

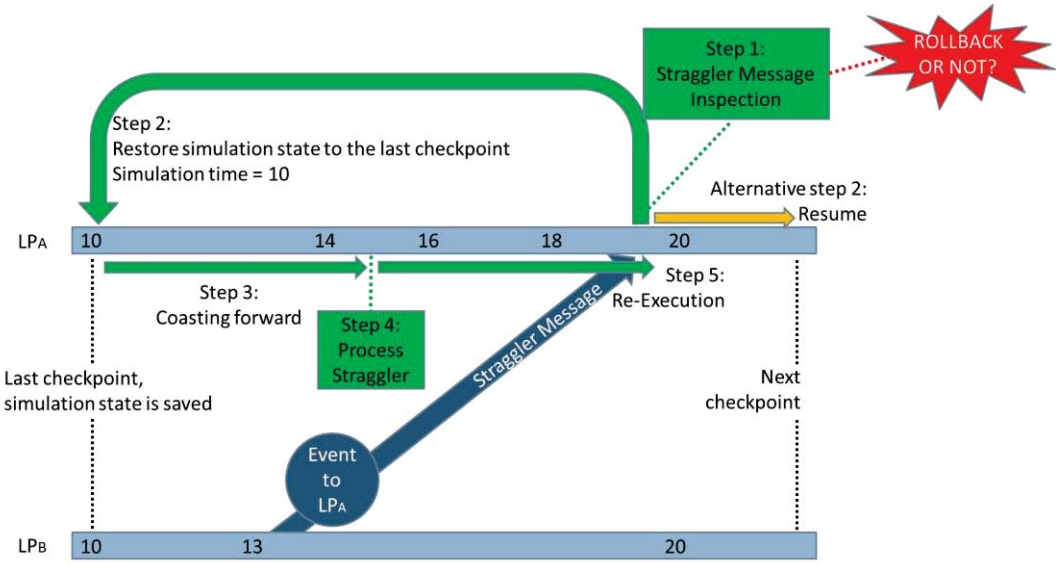

Figure 3: Sub-state saving algorithm.

In other words:

- If a rollback is required, sub-state saving method is C_{SMI} more expensive than conventional Time Warp.
- If a rollback is not required, sub-state saving method is cheaper than conventional Time Warp with the difference $C_{SR} + C_{CF} + C_{RE} + C_{AM} - C_{SMI}$. If straggler message inspection can be implemented as a light-weight method, sub-state saving method will worth employing.

4 IMPLEMENTATION STRATEGY FOR SPATIAL MODELS

We have implemented our method for agent-based models on a 2-dimensional spatial context. Before we present the implementation strategy, we briefly explain how RHPC_TW manages agents in space. RHPC_TW is the Time Warp extension of Repast HPC that is an open source, MPI based, distributed agent-based modeling and simulation tool (Collier and North, 2012). Repast HPC provides three types of context for developers: 2-dimensional grid, 2-dimensional continuous space and network. In Repast HPC, the workload is distributed to the LPs so that each is responsible for the same size of sub-grid as in Figure 4. In this example, there are nine LPs each managing a sub-grid that has 5x5 cells where agents live. An agent senses the other agents in its neighborhood and acts. If an agent is located in the buffer area, it can be a neighbor of an agent who belongs to another LP. Therefore, the agents in buffer areas should be visible to neighbor LPs at the right time. We have used RHPC_TW to simulate our spatial agent-based models. In the Time Warp algorithm, a rollback is performed if an LP receives the agents from a neighbor LP's buffer area at a later simulation time than it must receive. Perhaps, received agents would not cause an interaction even though they were received at the right time. Since they were not a part of any interaction, we call those agents as asocial agents. The sub-state saving method identifies the asocial agents. In spatial models, a sub-state refers to the states of the agents in the buffer. Since the set of those agents is a subset of the LP's full state, we call it a sub-state. Besides, a sub-state does not have to contain all attributes of agents, because some of them cannot be useful to determine whether a rollback is needed or not.

In our method, the interaction model should be provided by the user. Then, the simulation engine can find asocial agents and identify the straggler messages that can be ignored. To be more clear, we explain our method with an example. We have implemented Civil Violence model (Epstein 2002) in our experiments. In that model, there are four kind of agents:

- Cops who try to arrest activists,
- Activists who run away from cops,
- Quiets who wander around and calculate if they should rebel or not by considering activists and cops around,
- Jailed agents who cannot move for a specified time period.

An activist agent transforms to a jailed agent if it is arrested by a cop. After a while the jailed agent transforms to a quiet agent. If a quiet agent decides to rebel, it transforms to an activist agent. According to these principles of the Civil Violence model, the interaction model of agents is as follows:

- A cop has to see activists in its neighborhood.
- A quiet agent has to see cops and activists in its neighborhood.
- An activist has to see cops in its neighborhood.

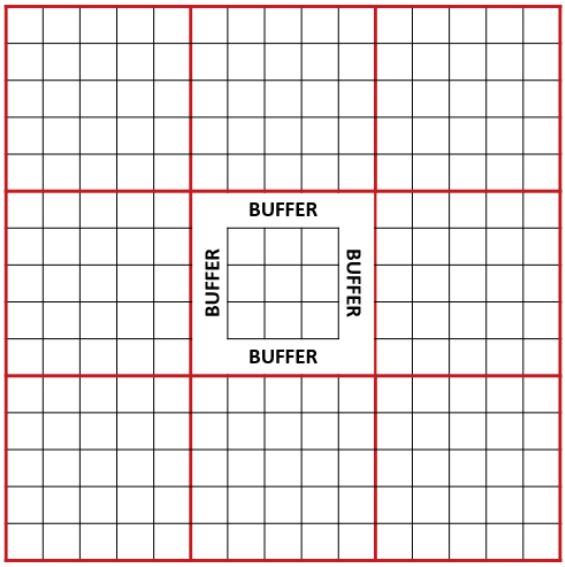

Figure 4: An example 2-dimensional grid in Repast HPC.

Therefore, LPs do not have to rollback when one of the interaction below appears:

- An activist did not see a quiet or a jailed agent at right time.
- A quiet agent did not see a jailed agent at right time.
- A jailed agent did not see any kind of agents at right time.
- A cop did not see a quiet or a jailed agent at right time.

Let's assume that LP_A and LP_B at time tick 10 is as in Figure 5 and the buffer size is 1, it means that an LP can see one row and one column from its neighbors. LP_B is notified by LP_A about Q (a quiet agent) and J (a jailed agent) when LP_B's local time is 15. In conventional Time Warp algorithm, LP_B does not know its past state at time tick 10, because that state had not been saved previously. So, LP_B has to perform a rollback operation by going back to the last checkpoint (assume it was at time tick 5). However, the straggler message would not cause an interaction at time tick 10, according to the interaction model above. Sub-state saving method handles this kind of cases. In sub-state saving method, LPs save some subsets that refer to the agents in buffer area, at every time tick. Saved sub-states allow LPs to examine straggler message and

make a decision whether the received agents are asocial or not. Now, LP_B can realize that it had only a C (a cop agent) agent in the buffer at time tick 10. So, LP_B identifies Q and J agents as asocial agents and avoid an expensive and unnecessary rollback.

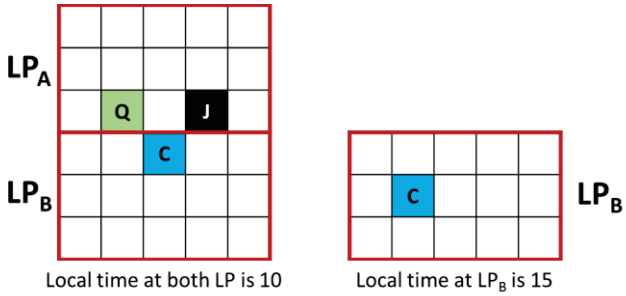

Figure 5: An example view of Civil Violence model.

The sub-state saving method has an overhead, but it is much smaller compared to the cost of saving full state. For instance, for an $N \times N$ sub-grid with the number of A agents and the buffer size of b;

- In full state saving, A agents should be saved with all attributes. Even the agents in a simple model may have many attributes. For instance, in the Civil Violence model, we have used 10 attributes (6 integer and 4 double) that include the location of the agent, agent type and other attributes specific to the model. In the GNU Compiler Collection (gcc) which we used, the sizes of integer and double variables are 4 bytes and 8 bytes, respectively. Therefore, the total size of a full-state is at least $A \times (6 \times 4 + 4 \times 8) = 56 \times A$ bytes.
- Through sub-state saving, a smaller number of agents will be saved. The number of buffer cells is $4 \times b \times (N\text{-}b)$. Therefore, there are $A \times (4 \times b \times (N\text{-}b) / N^2)$ agents in the buffer area on the average at any time. It means that $A \times (4 \times b \times (N\text{-}b) / N^2) \times (6 \times 4 + 4 \times 8) = 224 \times A \times b \times (N\text{-}b) / N^2$ bytes are saved at every time step on the average. Moreover, since only position (two variables of double) and type (one variable of integer) attributes of agents are saved in our case, size of the saved data will be $A \times (4 \times b \times (N\text{-}b) / N^2) \times (1 \times 4 + 2 \times 8) = 80 \times A \times b \times (N\text{-}b) / N^2$.

If the number of agents is 1000, the grid size is 1000 × 1000 and the buffer size is 1, then the full state saving method requires an LP to save 56000 bytes of data; while sub-state saving method requires an LP to save about 80 bytes of data on the average. This amount is the size of saved data at only one time step, but sub-states are saved at every time step. So, the proposed method actually requires $80 \times CheckpointInterval$ bytes of data to be saved between two checkpoints. The largest checkpoint interval size that we have seen in our experiments was 12. Therefore, total size of saved data with sub-state saving method is still much less than that with the full state saving method. This small size of data is sufficient to determine if a rollback is needed or not. Moreover, searching for which agents should be saved into a sub-state is not an overhead to the sub-state saving algorithm, because they are discovered during the buffer synchronization.

5 EXPERIMENTAL RESULTS

In the HPC system that we have used, there are 189 GB of memory and 4 AMD Opteron 6376 processors each of which has 16 cores and 2.3 GHz of clock frequency. The 40 of those cores are allowed to be employed at the same time. We have compared sub-state saving method with conventional Time Warp for the Civil Violence model on a 2-dimensional grid. We have experimented with that model with varying numbers of agents (12960 and 25920), buffer sizes (1, 2 and 3) and LPs (9, 16, 25 and 36) to observe which algorithm is better in which case. Finally, we adjusted checkpointing intervals dynamically with the additive increase multiplicative decrease method. The details of this checkpointing technique can be found in our earlier study (Görür et al. 2016).

Our method showed better performance than Time Warp in some cases. Figure 6 summarizes the results of our experiments. All of the experiments have been performed at least four times and the average of them is given here. We do not present the results for the state difference method, because they are clearly dominated by those for both conventional Time Warp and sub-state saving algorithm, as we explained in section 3.1. According to the results in Figure 6:

- When the simulation is performed with less number of LPs, conventional Time Warp algorithm shows better performance, because the overhead of sub-state saving method is considerable.
- A significant performance improvement in sub-state saving method was observed when larger number of LPs are used. Time Warp with sub-state saving catches up with the conventional Time Warp algorithm and beats it.
- Conventional Time Warp algorithm is hit by Amdahl's Law (Amdahl 1967), especially cores in different CPUs are employed. In all experiments with conventional Time Warp algorithm, using 25 LPs showed worse performance than using 16 LPs. However, employing sub-state saving method was not effected by Amdahl's Law in our experiments. Besides, the trend on the performance based on the number of LPs shows that using sub-state saving method can make conventional Time Warp algorithm more scalable when multiple CPUs are used.
- The buffer size is the same with the neighborhood range that is used in the model. Therefore, larger buffer size means more interaction potential among agents. Correspondingly, the execution times for both algorithms increase while the buffer size is increasing. Moreover, the rollback probability is directly proportional to the buffer size and number of agents. As the rollback probability increases, sub-state saving method shows better performance, because it can detect unnecessary rollbacks and avoid them. From this aspect, sub-state saving method can make Time Warp algorithm more scalable.

6 CONCLUSION

In this paper, we extended the conventional Time Warp algorithm with the sub-state saving method and compared it with the conventional Time Warp algorithm on a spatial model. An implementation strategy for agent-based models where agents live in a 2-dimensional grid is presented, as well. We introduced asocial agents that cause an LP to rollback in conventional Time Warp algorithm, although they do not interact with other agents. Through saving sub-states, we showed that getting rid of unnecessary rollbacks is possible by handling asocial agents.

Our experiments showed that sub-state saving method improves the performance of Time Warp, especially if the simulation is performed by large number of LPs. As the number of LPs increase, the execution time of simulation that employs sub-state saving method reduces much more than conventional Time Warp algorithm. Therefore, Time Warp algorithm can become more scalable by using sub-state saving method. Reduced number of rollbacks also brings an implicit advantage that improves the performance of Time Warp algorithm. Since our checkpointing algorithm adjusts the checkpoint interval considering rollbacks, the checkpoint intervals become larger when sub-state saving algorithm is employed comparing to conventional Time Warp algorithm.

Although sub-state saving method improves the performance of Time Warp algorithm, it comes with an overhead that cannot be ignored. This overhead is completely dependent on the buffer size and the number of agents in buffer area. As a future work, we are planning to reduce the cost of saving sub-states. We are also planning to experiment our method in other HPC systems that have more resources. We are going to compare the performance of our method with different models to investigate its generality.

ACKNOWLEDGEMENTS

We would like to thank the Department of Computer Engineering, Middle East Technical University, Ankara, Turkey, for the access provided to their parallel computing resources.

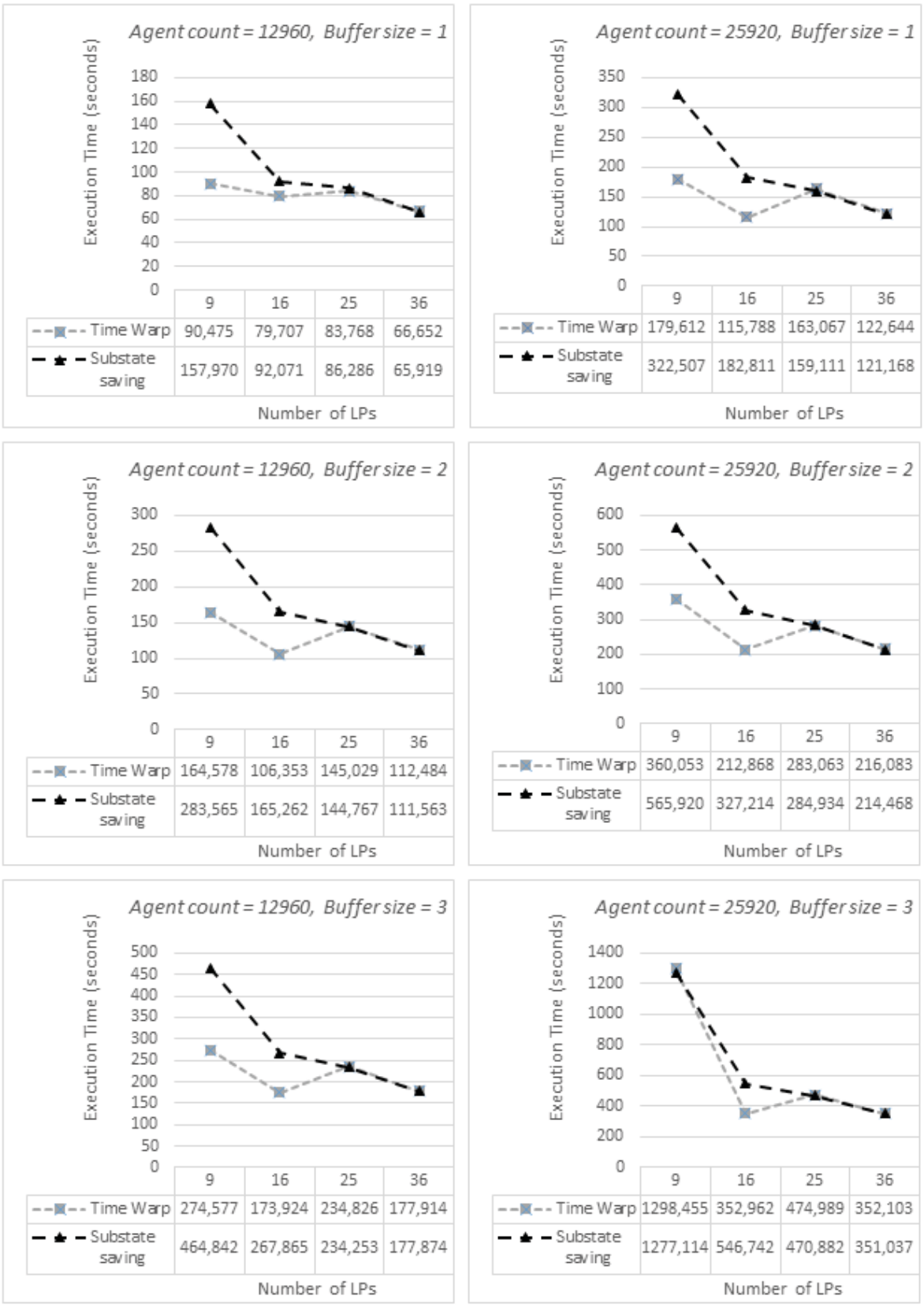

Figure 6: Performance comparison of conventional Time Warp algorithm and Time Warp algorithm with Substate Saving.

REFERENCES

Agrawal, D., and J. R. Agre. 1992. "Recovering from multiple process failures in the time warp mechanism". *IEEE Transactions on Computers* vol 41, pp. 1504–1514.

Amdahl, G. M. 1967. "Validity of the single processor approach to achieving large scale computing capabilities". In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pp. 483–485.

Barnes Jr, P. D., C. D. Carothers, D. R. Jefferson, and J. M. LaPre. 2013. "Warp Speed: Executing Time Warp on 1,966,080 Cores". In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pp. 327–336. Montreal, Quebec, Canada, ACM.

Bauer Jr, D. W., C. D. Carothers, and A. Holder. 2009. "Scalable Time Warp on Blue Gene Supercomputers". In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation,* pp. 35–44. IEEE Computer Society.

Collier, N., and North, M. 2012. "Parallel Agent-Based Simulation with Repast for High Performance Computing". *Simulation: Transactions of the Society for Modeling and Simulation International* vol 89, pp. 1215–1235.

Epstein, J. M. 2002. "Modeling Civil Violence: An Agent-Based Computational Approach". In *Proceedings of the National Academy of Sciences of the United States of America (PNAS)* vol 99, pp. 7243–7250.

Fachada, N., V. V. Lopes, R. C. Martins, and A. C. Rosa. 2016. "Parallelization Strategies for Spatial Agent-Based Models". *International Journal of Parallel Programming*, pp. 1–33.

Filatova, T., P. H. Verburg, D. C. Parker, and C. A. Stannard. 2013. "Spatial Agent-Based Models for Socio-Ecological Systems: Challenges and prospects". *Environmental Modelling & Software* vol. 45, pp. 1–7.

Fleischmann, J., and P. A. Wilsey. 1995. "Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators". *ACM SIGSIM Simulation Digest* vol 25, pp. 50–58.

Fujimoto, R. M. 1999. *Parallel and Distributed Simulation Systems*. New York, USA, John Wiley & Sons, Inc.

Glazer, D. W., and C. Tropper. 1993. "On Process Migration and Load Balancing in Time Warp". *IEEE Transactions on Parallel and Distributed Systems* vol 4, pp. 318–327.

Görür, B. K., K. İmre, H. Oğuztüzün, and L. Yilmaz. 2016. "Repast HPC with Optimistic Time Management". In *Proceedings of the 24th High Performance Computing Symposium (HPC' 16),* pp. 23–31. Pasadena, California, Society for Computer Simulation International.

Jafer, S., G. A. Wainer. 2010. "Conservative vs. Optimistic Parallel Simulation of DEVS and Cell-DEVS: A Comparative Study". In *Proceedings of the 2010 Summer Computer Simulation Conference (SCSC '10)*, pp. 342–349. Ottawa, Ontario, Canada, Society for Computer Simulation International.

Jefferson, D. R. 1985. "Virtual Time". *ACM Transactions on Programming Languages and Systems* vol 7, pp. 404–425.

Leong, H. V., D. Agrawal, and J. R. Agre. 1993. "Using Message Semantics to Reduce Rollback in the Time Warp Mechanism". In *Distributed Algorithms: 7th International Workshop, WDAG'93 Lausanne, Switzerland, September 27--29, 1993 Proceedings*, edited by Andrê Schiper, pp. 309–323. Berlin, Springer Berlin Heidelberg.

Lin, Y-B., B. R. Preiss, W. M. Loucks, and E. D. Lazowska. 1993. "Selecting the Checkpoint Interval in Time Warp Simulation". *SIGSIM Simulation Digest* vol 23, pp. 3–10.

Palaniswamy, A. C., and P. A. Wilsey. 1993. "An Analytical Comparison of Periodic Checkpointing and Incremental State Saving". *SIGSIM Simulation Digest* vol 23, pp. 127–134.

Perumalla, K. S. 2007. "Scaling Time Warp-based Discrete Event Execution to 10^4 Processors on a Blue Gene Supercomputer". In *Proceedings of the 4th International Conference on Computing Frontiers*, pp. 69–76. Ischia, Italy, ACM.

Presley, M. T., P. L. Reiher, and S. F. Bellenot. 1990. "A Time Warp Implementation of Sharks World". In *Proceedings of the 22nd Conference on Winter Simulation (WSC' 90)*, pp. 199–203.

Rönngren, R., and R. Ayani. 1994. "Adaptive Checkpointing in Time Warp". *ACM SIGSIM Simulation Digest* vol 24, pp. 110–117.

Thondugulam, N. V., D. M. Rao, R. Radhakrishnan, and P. A. Wilsey. 1999. "Relaxing Causal Constraints in PDES". In *Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. IPPS/SPDP 1999*, pp. 696–700.

Yajnik, S., and N. K. Jha. 1997. "Graceful Degradation In Algorithm-Based Fault Tolerant Multiprocessor Systems". *IEEE Transactions on Parallel and Distributed Systems* vol 8, pp. 137–153.

AUTHOR BIOGRAPHIES

B. KAAN GÖRÜR is a specialist computer engineer at Roketsan A.S. in Ankara, Turkey. He is also a PhD student in Department of Computer Engineering in Hacettepe University. His research interests are parallel and distributed simulation, agent based modeling and simulation, model driven engineering, and simulation visualization. His email address is bkaangorur@gmail.com.

KAYHAN İMRE is an assistant professor at the Department of Computer Engineering, Hacettepe University, Ankara, Turkey. He received his Ph.D. degree in Computer Science from University of Edinburgh, Scotland in 1993. His research interests are parallel computing, parallel and distributed simulation, real-time systems and photonic networks. His email address is ki@hacettepe.edu.tr

HALİT OĞUZTÜZÜN is a professor at the Department of Computer Engineering, Middle East Technical University (METU), Ankara, Turkey. He got his Ph.D. in Computer Science from University of Iowa, Iowa City, IA in 1992. His current research interests are model-driven engineering and distributed simulation. His email address is oguztuzn@ceng.metu.edu.tr

LEVENT YILMAZ is Professor of Computer Science and Software Engineering at Auburn University with a joint appointment in Industrial and Systems Engineering. He holds M.S. and Ph.D. degrees in Computer Science from Virginia Tech. His research interests are in agent-directed simulation, cognitive computing, and model-driven science and engineering for complex adaptive systems. He is the former Editor-in-Chief of Simulation: Transactions of the SCS and the founding organizer of the Agent–Directed Simulation Conference series. His email address is yilmaz@auburn.edu.