2015 The 5th International Workshop on Computer Science and Engineering
doi: 10.18178/wcse.2015.04.096

A Unconventional Rollback Synchronization Algorithm in Parallel and Distributed Simulation System

Xue-hui Wang[a], Lei Zhang[b]

State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China

[a]findyanzi@126.com, [b]zlmailbox2000@163.com

Keywords: parallel and distributed simulation(PDS), time management, synchronization, unconventional rollback algorithm.

Abstract. Simulation is a powerful tool for the analysis of new system designs, retrofits to existing systems and proposed changes to operating rules. In this paper we limit our discussion to parallel and distributed simulation (PDS). In order to simulate large-scale complex systems with better consistency, further more as fast as possible, the universally adoptive approach of PDS is that make the execution of simulation programs on multiprocessor and distributed computing platforms. Time management algorithm is one of the key techniques in the parallel and distributed system simulation, which broadly fall into conservative and optimistic synchronization. A survey of both the two algorithms is presented focusing on fundamental principles and mechanisms. The remainder of this paper is focused on one of the novel rollback algorithm; we call it as unconventional rollback algorithm. And then, we provide and describe the novel rollback algorithm in optimistic time management in detail, including scheduler's priority queue, rollback manager, cancellation strategies, and roll forward operation. Among this paper, central issues concern the synchronization of computations on different processors. Finally, we discuss how to get the relatively minimal rollback, and how to realize the dynamic allocation and reclamation.

Introduction

In this paper, we first give an overview of Parallel and Distributed Simulation (PDS), in which involves the advantage and the architecture of PDS. Afterwards two main approaches in synchronizing parallel simulation execution, briefly either conservative or optimistic are reviewed. Then, in respect that rollback algorithm adopted in the optimistic approach may potentially uncover a higher degree of parallelism in the simulated system; we discuss the traditional rollback mechanism and set forth a novel rollback algorithm, termed as unconventional rollback. Later we provide an emphatic study to describe the novel rollback algorithm in optimistic time management in detail, including scheduler's priority queue, rollback manager, cancellation strategies, and roll forward operation. Sequentially we discuss how to get the relatively minimal rollback, and how to realize the dynamic allocation and reclamation in the rollback.

Traditional Rollback Mechanism

The Time Warp mechanism (Jefferson 1985) is the most well known optimistic method. When an LP receives an event with timestamp smaller than one or more events it has already processed, it rolls back and reprocesses those events in timestamp order. Rolling back an event involves restoring the state of the LP to that which existed prior to processing the event, and "cancel-sending" messages sent by the rolled back events. An elegant mechanism called anti-messages is provided to "cancel-send" messages.

An anti-message is a duplicate copy of a previously sent message. Whenever an anti-message and its matching (positive) message are both stored in the same queue, the two are deleted. To cancel sending the message, a process need only send the corresponding anti-message. If the matching

positive message has already been processed, the receiver process is rolled back, possibly producing additional anti-messages. Using this recursive procedure all effects of the erroneous message will eventually be erased.

There are still two problems remain to be solved. For one thing, certain computations, e.g., I/O operations, cannot be rolled back. For another thing, the computation will continually consume more and more memory resources because a history must be retained, even if no rollbacks occur; some mechanism is required to reclaim the memory used for this history information. Both problems are solved by global virtual time (GVT). [6] GVT is a lower bound on the timestamp of any future rollback. GVT is computed by observing that rollbacks are caused by messages arriving "in the past." Therefore, the smallest timestamp among unprocessed and partially processed messages gives a value for GVT. Once GVT has been computed, I/O operations occurring at simulated times older than GVT can be committed, and storage older than GVT (except one state vector for each LP) can be reclaimed.

Unconventional Rollback Algorithm

In order to avoid an explosion of cascading anti-messages and get relatively high-powered parallel and distributed simulation, we provide a novel rollback algorithm, termed as unconventional rollback algorithm. Simulation object manage both their set of unprocessed pending events and their processed but uncommitted events. Each event has a Rollback Manager that stores Specialized State Altering Items (SSAI) items that are created as capable-rollback operations are performed while processing the event. SSAI items undo the operation that generated them when rollbacks occur.

Schedule priority queue. Each node in our simulation drive provides a scheduler that coordinates event processing for its local simulation objects. The scheduler maintains each simulation object in a priority queue using each simulation object's next unprocessed event time as its priority. Priority queues have two primary operations: insert and remove. The insert operation inserts an item with a priority value (e.g., a time tag). The remove operation removes the item with the highest priority (e.g., the smallest time tag). The same priority queue data structure can be used to manage simulation object in the scheduler or events in a simulation object. The scheduler first removes the simulation object with the earliest unprocessed event time. It then processes the simulation object's next event. Afterwards, the scheduler inserts the now processed event into the simulation object's optimistically processed event list. Finally, the scheduler reinserts the simulation object back into its priority queue using the simulation object's next unprocessed event time as its priority. If the simulation object has no more pending events, then it does not need to be inserted in the scheduler's priority queue. It is possible for another simulation object to later schedule an event for a simulation object that has no pending events. When this happens, the scheduler will insert the simulation object back into its priority queue to coordinate its event processing.

Rollback Manager. To accomplish this, each simulation object maintains its set of pending events in its own priority queue. In addition, each simulation object also manages its optimistically processed but uncommitted events in a doubly linked list. The optimistically processed list of events is cleaned up when Global Virtual Time (GVT) is determined. Events with time tags less than GVT are committed and then removed from the list of optimistically processed events.

Event objects contain a Rollback Manager, which stores SSAI that are created when capable-rollback operations are performed. The SSAI base class defines two virtual functions: Rollback() and Cleanup(). These functions are used to implement support for rollbacks. Each state-saving operation generates an instance of a derived SSAI class to uniquely implement those two virtual functions for the operation. The specialized SSAI classes are inserted into the event's SSAI Monitor as they are created. The event's SSAI Monitor then provides a stack of specialized SSAI items. These SSAI items are responsible for undoing the operations that caused their creation in case rollbacks occur. The Rollback() virtual function is invoked on each SSAI in reverse order to restore the simulation object back to its original state before the event was processed.

Cancellation Strategiesk Manager. When a rollback occurs, some output messages may need to be cancelled. There are two categories of cancellation strategy—aggressive and lazy.

Aggressive Cancellation: When the aggressive cancellation strategy is used, antimessages are sent immediately when a rollback occurs. Such messages often lead to secondary rollbacks in other LPs. The assumption is that the cancelled messages may be causing erroneous computation in other LPs.

Lazy Cancellation: When the lazy cancellation strategy is used, antimessages are not sent when a rollback occurs. Instead, antimessages are placed into a queue of pending antimessages. When the LP resumes execution, it will generate output messages. In the event that an output message is the same as a message that would have been cancelled during the rollback, then the pending antimessage and the new output message will annihilate.

The assumption is that after a rollback, an LP is likely to produce the same output messages. In this case, the lazy cancellation strategy will reduce the unnecessary secondary rollbacks that would occur with aggressive cancellation.

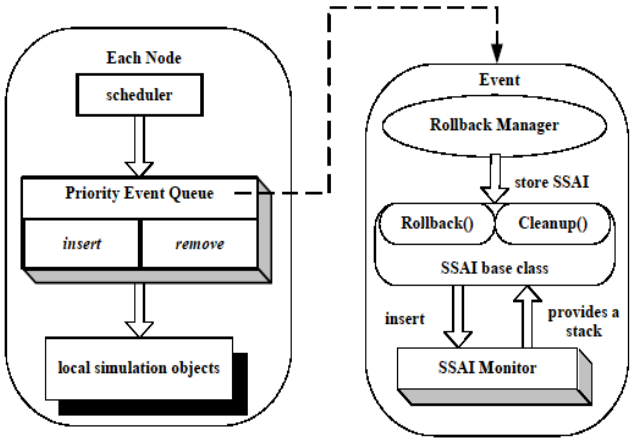

Fig.1 the Relationship between the Node, Simulation object, Event, SSAI Monitor and SSAI base class

Roll Forward Operation. Sometimes, events that have been rolled back do not need to reprocessed, but instead can be rolled forward. This occurs when a straggler does not modify any of the state variables that the rolled-back event depends on. This performance enhancing technique is called Lazy Cancellation because the rolled-back event waits and tries to roll forward before cancelling the events it may have generated. To support roll forward operations, each SSAI item must roll forward whenever the Rollback() virtual function is called again. Two rollbacks on an event are equivalent to a rollback followed by a roll forward. In the most general sense, an odd number of rollbacks restore the event back to its state before the event was processed. While an even number of rollbacks restores the simulation object back to its state after the event was originally processed.

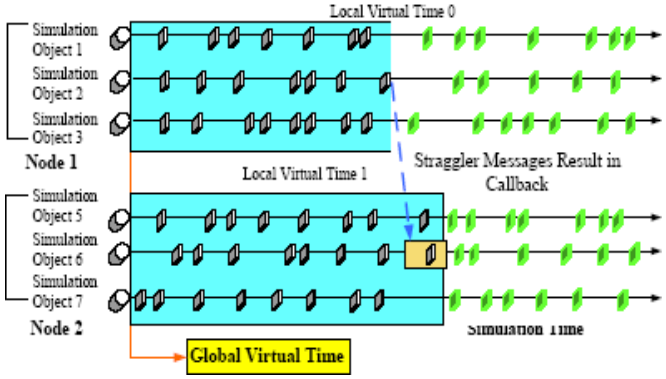

Fig.2 straggler messages arrive from other node cause rollback

When a node of parallel simulation receives a straggler message, it does not rollback all of the locally processed events that have a greater time value. Instead, it only rolls back the necessary events that were processed by the target simulation object. This is shown in Fig.2.

More parallelism is achieved through limiting rollbacks in this manner. This approach results in fewer rollbacks and better overall parallel performance. However, it does mean that events must be associated with one and only one, simulation object. Accessing and/or modifying the state variables of another simulation object in an event are forbidden.

Dynamic Allocation and Reclamation. Often event processing requires the allocation and/or reclamation of memory. In the case of allocation this must be done in a reclamation manner to avoid memory leaks, and also access to freed memory at reclamation. We provide the DYNMEMORY__CLASS (N) macro to support reclamation dynamic memory operations. The code how to use this macro is provided below:

```
/*an example how to use the macro DYNMEMORY _CLASS */
#define AppClassGate
#include “RollBack.H”//Fundamental data types
AppClass {
public:
AppClass() {}
AppClass(int a) {B = a;}
int GetB() {return B;}
void SetB(int a) {B = a;}
private:
RollBack_int B;
};
/*the macro defines rollback memory allocation and reclamation functions for AppClass*/
DYNMEMORY_ CLASS (AppClass)
#endif
```

The T argument is the name of the class or type being defined for rollback dynamic memory operations. Please take notice of the importance to clearly know that the DYNMEMORY_ CLASS macro does not make the internals of the class rollback. It only provides rollback memory allocation and reclamation operations. This macro is normally after the placed class definition in its header file and defines several important functions and unconventional pointers.

Conclusion

Parallel simulation is concerned with the execution of simulation programs on multiprocessor computing platforms. While distributed simulation is concerned with the execution of simulations on geographically distributed computers interconnected via a local area and/or wide area network. Parallel and distributed simulation technologies address issues concerning the execution of simulation programs on multiprocessor and distributed computing platforms.

Time management algorithm is one of the key techniques in the parallel and distributed system simulation. Much of the work concerning the execution of analytic simulations on multiprocessor computers is concerned with synchronization. The synchronization algorithm ensures that the cause and effect relationship in the system being simulated are correctly reproduced in the simulation program. Synchronization in time management is a well-studied area of research in the parallel and distributed system simulation field.

In our paper, we first give an overview of Parallel and Distributed Simulation (PDS). Then a survey of both the two algorithms is relatively presented focusing on fundamental principles and mechanisms. Afterwards, in respect that rollback algorithm adopted in the optimistic approach may potentially uncover a higher degree of parallelism in the simulated system; we simply discuss the

traditional rollback mechanism and set forth a novel rollback algorithm, termed as unconventional rollback, in the remainder of this paper. Later we provide and describe this unconventional rollback algorithm in detail. Such as scheduler's priority queue, rollback manager, cancellation strategies, and roll forward operation, even minimal rollback and the dynamic allocation and reclamation all are discussed.

References

[1] Jefferson D. Virtual Time. [J]. ACM Transactions on Programming Languages and Systems, 1985. Vol. 7(3), p. 404-425.

[2] Li, Cheng-Hong; Park, Alfred J.Analytical performance modeling for null message-based parallel discrete event simulation. Proceedings of MASCOTS 2011. p349-358.

[3] Liu, Jason. Real-time scheduling of logical processes for parallel discrete-event simulation. Proceedings of the 2013 Winter Simulation Conference. 2013, p2959-2971

[4] xuehui Wang, Research on the Time Management Technology in Parallel and Distributed Simulation Systems [D], National University of Defense Technology, 2006.

[5] De Munck, S.; Vanmechelen, K.; Broeckhove, J. Revisiting conservative time synchronization protocols in parallel and distributed simulation. CONCURRENCY AND COMPUTATION-PRACTICE & EXPERIENCE Vol.26(2) P.468-490, feb.2014.

[6] R.M. Fujimoto. Parallel discrete event simulation：Will the field survive？ORSA Journal of Computing, 5(3):213-230, 2005, 17(11): 2727-2730.