HYBRID SYNCHRONIZATION IN THE DCB BASED ON UNCOORDINATED CHECKPOINTS

Flávio Marques Migowski Carvalho
Agência Nacional da Aviação Civil
Distrito Federal, Brasília, Brasil
email: fmigowski@gmail.com

Braulio Adriano de Mello
Universidade Federal da Fronteira Sul
Santa Catarina, Chapecó, Brasil
email: braulio@uffs.edu.br

KEYWORDS
Uncoordinated Checkpoint, Hybrid Synchronization, DCB, Rollback.

ABSTRACT

Hybrid timing synchronization models that combine synchronous and asynchronous components are error prone while executing rollback operations. Mechanisms such as checkpoints are able to guarantee the correctness of these models when the rollback operations of asynchronous components are unavoidable. Checkpoints are saved states of the components in order to recover a previous and consistent state in case of violations of the time constraints. This work presents the specifications and the implementation of uncoordinated checkpoints for supporting rollback operations of asynchronous components based on the DCB architecture. Uncoordinated checkpoints allow us to avoid low performance of the simulations by reducing the overhead of control messages.

INTRODUCTION

In distributed simulations, the different parts of the model execute distinct sequences of events and they do not have their local times synchronized. As each event of each component is able to schedule new events which send new messages to other components, then the synchronization mechanisms are needed to deal with violations of Local Causality Constraints (LCC). One of the issues is the management of checkpoints to execute rollback in case of LCC violations into the asynchronous parts. The recovery from timestamp order violations could be done by a checkpoint-based rollback.

There are studies to improve the management of checkpoints, for example, by reducing the checkpoint time overheads (Soliman and Elmaghraby 2002), and selecting the best positions for checkpoints instead of taking them on a periodic basis (Quaglia 2001), and recovering (Cortellessa and Quaglia 2001) by using non-blocking checkpoints (Sato et al. 2012) to improve the eciency or working with incremental checkpoints to reduce the overhead (Feng and Lee 2006). The management of checkpoints is more complex when the system combines synchronous and asynchronous components (hybrid systems). Solutions such as the mapping of the conservative HLA synchronization interface onto the optimistic one (Santoro and Quaglia 2012) work on the interface of the components. Due to the complexity of real-world systems, more recent studies have proposed solutions to deal with distributed and heterogeneous systems (Reynolds 1988) based on checkpoints and rollbacks for failure detection and failure recovery (Malensek et al. 2013) in distributed simulations.

Besides these issues, heterogeneous models frequently combine synchronous and asynchronous components. As synchronous components cannot rollback their local time, then the simulator must integrate functionalities to keep the model consistent even in the case of rollback of the asynchronous parts. This goal is more complex when the model has components with diversity of modeling languages and their functional interfaces do not match directly to each other.

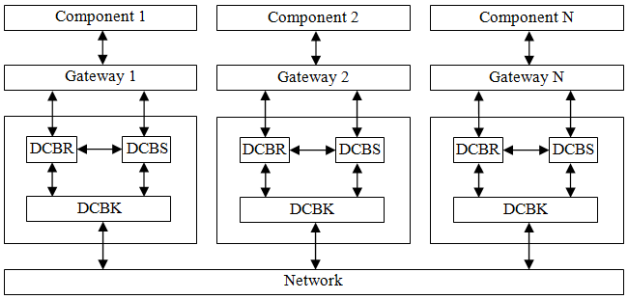
This work presents the integration of functionalities into the DCB (Distributed Cosimulation Backbone) (Mello et al. 2005) to manage rollback operations based on an uncoordinated checkpoints approach. According to the features of the DCB architecture to support hybrid models, the integration of skills to manage uncoordinated checkpoints is mandatory to guarantee the independence of the heterogeneous components. Besides the susceptibility to the domino effect and to the orphan messages, the checkpoint management strategies integrated into the DCB are able to minimize these effects by reducing the number of control messages drastically.

In the remainder of the paper the Background section presents the main issues of DCB and rollback strategies. The specification of the functionalities, implementation of uncoordinated checkpoints into the DCB and results are presented in the next sections. Following are the conclusions and then references are presented.

BACKGROUND

The DCB (Distributed Cosimulation Backbone) (Mello et al. 2005) is a simulation architecture for heterogeneous discrete simulation model. It offers resources for cooperation of distributed and heterogeneous components preserving the original code and internal configuration of the components.

The DCB encapsulates adaptation layers for communication and synchronization among heterogeneous and remote components, so that the code for components becomes completely independent from the DCB communication mechanisms. Four main modules, as shown in Figure 1, execute the management of the cooperation process among the model components. They are: DCBS (DCBSender), DCBR (DCBReceiver), DCBK (DCBKernel), and Gateway.



Figures 1: DCB Architecture

There are four main modules in the DCB architecture: DCB Receiver (DCBR), DCB Sender (DCBS), DCB Kernel (DCBK) and the gateway. The main gateway task is the interface translation of components. It provides an interface to the other components and it offers data translation services for supporting incompatible interfaces due to the heterogeneity. The DCBR handles the messages management received from other components. The DCBS manages the sending of messages generated in the local element. DCBS and DCBR work together to govern the Local Virtual Time (LVT) of elements. The DCBK manages the message exchange services and the Global Virtual Time (GVT).

All modules of the DCB execute only local operations except for the DCBK. It manages the local or remote connections with the other components based on the configuration sets. All those modules work in a cooperative way to guarantee the correctness of the simulation. In addition, models with hybrid synchronization are supported by the DCB architecture. Its functionalities allow combining synchronous and asynchronous components. Synchronous components are not allowed to execute rollback operations.

Since the generation of the configuration files is an onerous task, the DCB integrates a module for automatic generation of the components configuration for their integration into heterogeneous and distributed models. This module is split in two main parts: the distributed repository; and the automatic configuration of models. The first part, a distributed repository manager of components, offers location transparency of components and automatic collaboration among distributed repositories to broadcast components search actions. The second part allows the automatic configuration of input and output ports of the components and the interface translation when the sender and receiver components have heterogeneous interfaces. The new functionalities for supporting rollback operations presented in this work are compliant with the current DCB functionalities.

Rollback based on checkpoints

Rollback operations must be executed in case of violations of LCC (Local Causality Constraint) by one or more components (Elnozahy et al. 2002) and they can affect only asynchronous components. There are two main methods to execute rollback operations: checkpoint-based and log-based (Koo and Toueg 1986).

The log-based rollback could be supported by optimistic protocols or pessimistic protocols. The optimistic protocols allows for the reducing of the control messages overhead (Valchanov et al. 2008). However, they do not avoid the orphan messages thereby increasing the processing load of the garbage collection operations. The pessimistic protocols allow us to avoid orphan messages, however they increase the overhead. Some studies propose solutions which combine optimistic and pessimistic protocols. These studies make efforts to integrate the positive aspects of both protocols but they do not have a stronger effectiveness (Soliman and Elmaghraby 2002).

Protocols based on a checkpoint have three main categories: uncoordinated checkpointing; coordinated checkpointing; and communication-induced checkpointing. In the uncoordinated protocols approach, each component creates its own checkpoints independently. This feature is important to keep the independence among components according to the DCB architectural features. In fact, besides the complexity to treat the domino effect (Koo and Toueg 1986) and that some of the checkpoints could not be part of a global consistent state, the uncoordinated checkpoint approach is better than other approaches to integrate functionalities in the DCB for supporting rollback operations. The main reason is that the capacity to reduce the overhead is a great advantage for distributed models.

Coordinate checkpoints (Moreira et al. 2005) require the processes to organize the checkpoints of all components to compose a consistent global state. They are not susceptible to the domino effect. However, they increase the overhead due to the control messages. The overhead is the main disadvantage. Although the coordination

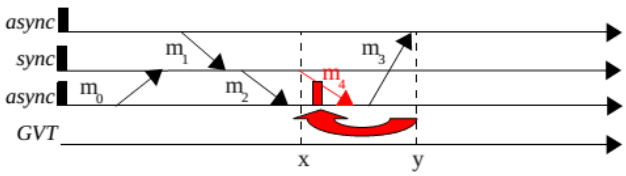
requires more cooperation among all the components of the model to generate secure checkpoints, it make the rollback operations simple since all processes return only to the most recent checkpoint in case of rollback. The most recent checkpoint is always enough to get back the consistency of the simulation. The oldest checkpoints can be discarded.

The communication-induced checkpoints allow for avoiding the domino effect without coordination among all the components. In order to coordinate the checkpoints, the components create local or forced checkpoints. The local checkpoints are independent and they are not a part of a consistent global state. A guaranteed global time line is mandatory for creating the forced checkpoints. They use information of the communication messages of components to guarantee the consistent global state.

MANAGEMENT OF CHECKPOINTS BY THE DCB

An uncoordinated checkpoints approach was chosen to be integrated into the DCB architecture due to the following DCB features: components are completely independent (they do not communicate directly with each other); DCB supports heterogeneous and distributed models; and the interoperability among different modeling languages. In spite of the uncoordinated checkpoints being susceptible to the domino effect and requiring garbage collection, the cost of the coordinated protocols could be excessively onerous on distributed and heterogeneous models.

The DCB is responsible for guaranteeing the consistency of the checkpoints based on the information from the communication messages and to manage the rollback operations in case of LCC violation. Moreover, asynchronous components must be able to create their own internal checkpoints to recover previous internal states in case of rollback operations. Currently, the DCB does not integrate functionalities to identify safety states in order to create checkpoints only in consistent global states. This goal is seen as future work. These functionalities will be useful to reduce the impact of the domino effect and the impact of the garbage collection operations.



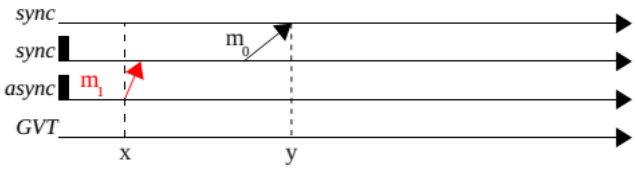
Figures 2: Example of worthless checkpoint

There is the need to confirm whether a state is safe or unsafe before deciding whether a checkpoint is consistent or not. Figure 2 illustrates a scenario where a synchronous component with the local time equal to x sends the message m_4 to an asynchronous component with the local time equal to y which is greater than the timestamp of m_4 (LVT > m_4(t)).

An arrival of the message m_4 with a timestamp less than the LVT of the receiver component, illustrated in Figure 2, works as a trigger for the third component to come back to the first checkpoint before the timestamp of m_4. As the checkpoint is before the sending time of the message m_3, then m_3 is discarded. Due to the discarding of m_3 then the first component comes back to the previous checkpoint which is at the beginning of the simulation in this example. These actions also require the discarding of the message m_1. This example shows a domino effect scenario where all components are triggered for rollbacking until the simulation time of zero.

In the same scenario illustrated in Figure 2, the second checkpoint of the third component created after the local time x is worthless and it can be discarded by the DCB without affect to the simulation.

Figure 3 illustrates an inconsistent state where an asynchronous component sends the message m_1 to a synchronous component whose LVT is greater than the timestamp of m_1. Models with hybrid synchronization do not accept rollback operations for synchronous components. When a synchronous component receives a message with a timestamp less than its current time, the simulation is considered to be failed. Due to that, the DCB takes into account only the LVT of the asynchronous components plus the lookahead to update the GVT (Global Virtual Time) (Fu et al. 2013).



Figures 3: Example of inconsistent state

As the DCB takes into account only the LVT of the asynchronous components for updating the GVT, then the rollback operations will never cause reducing of the GVT. Due to that, the synchronous components always keep their LVT consistent in relation to the GVT. When a synchronous component receives a message with a timestamp less than the LVT then the DCB sees a deadlock and the simulation is canceled. The internal functionalities of the DCB which manage the rollback operations also works to prevent the deadlocks.

Uncoordinated checkpoints allows each component to build checkpoints at distinct LVT. There are two main advantages: reducing of the overhead caused by control messages to coordinate the checkpoints; and the support of the DCB features to keep the independence of the components. The coordinated checkpoints require

specific functionalities to manage the time advance of the LVT of each component in a coordinated way. These functionalities are not in compliance with the above advantages mainly for distributed and heterogeneous models.

DCB functionalities for rollback supporting

The rollback procedures are encapsulated by the DCBS and DCBR modules. They are responsible for managing the received and sent messages by the components. Each component has its own copy of these modules and its own gateway. These modules and the gateway are responsible for communicating processes among the components based on the input and output ports configuration. Whenever the gateway of a component identify an anti-message, then the timestamp of the anti-message is used to verify whether there are messages in the local buffer which must be discarded according to the rollback process. This operation is executed by the DCBR.

In Figure 4 there is presented the simplified code of the gateway functions to manage the rollback operations. Whenever there are messages in the local buffer with the timestamp less than the timestamp of a recent received anti-message, the DCBS sends new anti-messages to the components which also need to execute the rollback operation. The global rollback finishes when the last consistent checkpoint is reached and there are no more anti-messages to be sent by the components.

Line 4 of Figure 4 shows whether the component is processing the rollback operations or not. If true, the LVT advance is prohibited while the component does not finish the rollback execution and restarts the simulation advance.

```
1.      A0 = DCBR.getAttributeReceived("444.3");
2.      if (A0 != null) {
3.         DCBR.AttributeRemove(A0);
4.         Fed.rollback = true;
5.         checkpoint = Fed.getCheckpoint(A0.LVT);
6.         if ( checkpoint != null) {
7.            DCBS.antiMessageTrigger(checkpoint);
8.            Fed.rollback(checkpoint);
9.            Fed.setChatLVT(updateLVT(A0.LVT));
10.           Fed.setReceivedText(A0.Value);
11.           Fed.rollback = false;
12.        }
13.     }
```

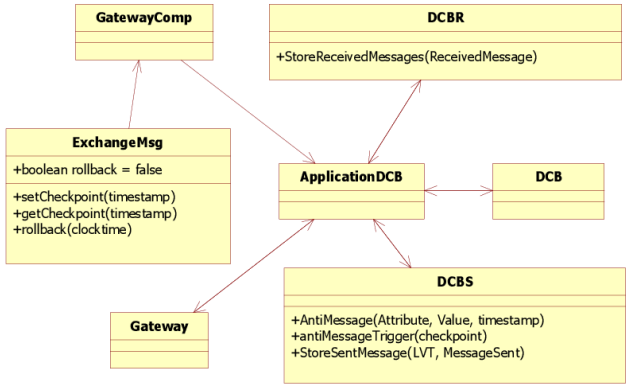
Figures 4: Rollback functions of DCB

Line 5 of the gateway shows the functionality responsible for finding the checkpoint with the greater time less than the timestamp of the anti-message (last consistent checkpoint). When the checkpoint is identified then the rollback operation continues by sending new anti-messages according to the new checkpoint if necessary (line 8) followed by the LVT update (line 9) and the receiving of the message which started the rollback operation (line 10).

The simplified class diagram in Figure 5 shows the main methods which implement the new functionalities for supporting the uncoordinated checkpoints into the current version of the DCB. The ExchangeMsg class belongs to the case study presented in this work.

The DCBS keeps the messages sent by the component in a local buffer in order to manage receiving anti-messages in case of rollback operations. The DCBR logs the history of sent and received messages for supporting the gateway to communicate with the local component.

In Figure 5 the component implemented by the class *ExchangeMsg* is able to create new checkpoints using the method *setCheckpoint()* and to execute the rollback operations based on previous checkpoints. As the DCB is using uncoordinated checkpoints, the components have no time constraints to define the states on the time line to create new checkpoints.



Figures 5: DCB structure for supporting uncoordinated checkpoints

Asynchronous components are also required to be able to manage internal rollback operations. As the DCB communicates with the components only by the communication ports of the interface, then the internal processing of the components is not affected by the functionalities of the DCB. The DCB manages the exchange of messages and anti-messages but it does not manage the internal processing of the components. If a component does not create internal checkpoints with information about its internal processing, it must be integrated into a model as a synchronous component. In general, asynchronous components implement internal procedures to process anti-messages in case of rollback.

IMPLEMENTATION AND RESULTS

We integrated the proposed uncoordinated checkpoints solution into the current version of the DCB and we implemented a simple heterogeneous model for validating purposes. The model represents a message

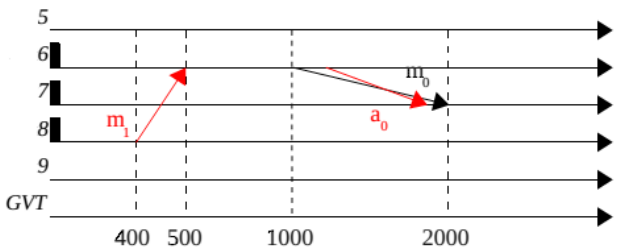
exchange system (ExchangeMsg) and it integrates synchronous and asynchronous components. The following components were created for the case study:

• C5 and C9 (synchronous)
• C6, C7 and C8 (asynchronous)

All the components have the same internal behavior, they are independent and they communicate among themselves for exchanging simple text messages. The simulation time *0* is seen by the DCB as the first checkpoint and it is the lowest limit to the rollback operations. The DCB generates a new checkpoint for each component in period intervals of ten (10) received or/and sent messages. The current LVT less than the LVT of the last checkpoint must be equal to or less than 5000 time units. We have not applied strategies to calculate the best intervals between the checkpoints because the main goal of this case study is to validate the rollback operations. So, it was enough to generate states with a time constraints violation for asynchronous components for validating purposes. The estimation of the best intervals between checkpoints is seen as future work.

During the execution of the simulation the DCB identified messages received by the asynchronous components whose LVT were greater than the timestamp of the messages (LCC violation). One of these situations is illustrated in Figure 6 and it has the following sequence of steps:

• LVT(C6) = 1000 (LVT of Component 6 is equal to 1000) and C6 sends the m_0(2000) (message m_0 with timestamp equal to 2000) to C7;
• When LVT(C7) = 2000 then C7 receives m_0 from C6;
• LVT(C8) = 400 when C8 sends m_1(500) to C6;
• DCB detect LCC violation and it sends a(500) (anti-message with timestamp 500) to C6 ;
• Then the Gateway of C6 starts the rollback process.



Figures 6: Scenario of imminent rollback

After sending the anti-messages and having started the rollback process, the following steps are executed:\\

• The LVT advancing of C6 is interrupted;
• DCBS of the C6 sends a_0(1999) to C7.

Then, C7 discards the message m_0 sent by C6 and returns to the nearest checkpoint before the current LVT. After the rollback processing, C6 restarts the time advance process and it receives the message m_1(500) sent by C8.

Whenever an asynchronous component with *LVT(t)* sends a message to a synchronous component then the *LVT(t+1)* becomes the new lowest limit for rollback operations. This restriction is required to guarantee the consistency of the synchronous components. This functionality is integrated into the DCB. In this scenario, when an asynchronous component needs to return to a state before *LVT(t+1)* then the simulation is considered to be failed and the model must be reviewed. These issues are foreseen in the new case studies.

CONCLUSIONS

This work presented a set of functionalities for supporting rollback operations by the DCB of hybrid timing synchronization models based on an uncoordinated checkpoints approach. These functionalities enable the DCB to create uncoordinated checkpoints and to manage rollback operations keeping the time correctness of the components.
We chose an uncoordinated checkpoints approach in compliance with the main DCB features about the independence of the components, the supporting for heterogeneous and distributed models, and the interoperability among different modeling languages.

In our proposed solution the components are free to define their own checkpoints without concerning the timing coordination with the other components. This feature allows for the reducing of the management messages and guarantees the correctness of the simulation. The future works include the specification of strategies to reduce the creation of worthless checkpoints, and the definition of strategies to find the best intervals between two checkpoints based on induced-techniques and information collected from the simulation messages, and performance analysis between scenarios based on uncoordinated and coordinated checkpoints.

Acknowledgments
Braulio Adriano de Mello acknowledges the support from the Brazilian research funding agency CAPES (Coordenaçao de Aperfeiçoamento de Pessoal de Nível Superior), proc n. 1835-14-9.

REFERENCES

Cortellessa V. and Quaglia F., 2001. "A checkpointing-recovery scheme for time warp parallel simulation". *Parallel Computing*, 27, no. 9, 12261252.

Elnozahy E.N.M.; Alvisi L.; Wang Y.; and Johnson D.B., 2002. "A survey of rollback-recovery protocols in message-passing systems*". ACM Comput Surv*, 34, no. 3, 375408.

Feng T.H. and Lee E.A., 2006. "Incremental Checkpointing with Application to Distributed Discrete Event

Simulation". *In Proceedings of the 38th Conference on Winter Simulation*. Winter Simulation Conference, 1004-1011.

Fu D.; Becker M.; and Szczerbicka H., 2013. "On the Potential of Semi-conservative Look-ahead Estimation in Approximative Distributed Discrete Event Simulation". *In Proceedings of the 2013 Summer Computer Simulation Conference*. Society for Modeling & Simulation International, 28:128:8.

Koo R. and Toueg S., 1986. "Checkpointing and Rollback-recovery for Distributed Systems". *In Proceedings of 1986 ACM Fall Joint Computer Conference*. IEEE Computer Society Press, 11501158.

Malensek M.; Sui Z.; Harvey N.; and Pallickara S., 2013. "Autonomous, Failure-resilient Orchestration of Distributed Discrete Event Simulations". *In Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference.* ACM, 3:13:10.

Mello B.A.; Souza U.R.F.; Sperb J.K.; and Wagner F.R., 2005. "Tangram: Virtual Integration of IP Components in a Distributed Cosimulation". *IEEE Design and Test*, 22, no. 5, 462471.

Moreira E.M.; Helena R.; Santana C.; and Santana M.J., 2005. "Using Consistent Global Checkpoints to Synchronize Processes in Distributed Simulation". *In Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications*. IEEE Computer Society, 4350.

Quaglia F., 2001. "A cost model for selecting checkpoint positions in time warp parallel simulation". *IEEE Transactions on Parallel and Distributed Systems*, 12, no. 4, 346 362.

Reynolds P.F., 1988. "Heterogeneous Distributed Simulation". *In Proceedings of 1998 Winter Simulation Conference*. University of Virginia, 206209.

Santoro A. and Quaglia F., 2012. "Transparent optimistic synchronization in the high-level architecture via time-management conversion". *ACM Transaction on Model Computer Simulation*, 22, no. 4, 21:121:26.

Sato K.; Maruyama N.; Mohror K.; Moody A.; Gamblin T.; de Supinski B.R.; and Matsuoka S., 2012. "Design and Modeling of a Non-blocking Checkpointing System". *In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 19:119:10.

Soliman H.M. and Elmaghraby A.S., 2002. "An analytical model for hybrid checkpointing in time warp distributed simulation". *IEEE Transactions on Parallel and Distributed Systems*, 9, no. 10, 947951.

Valchanov H.; Ruskova N.; and Ruskov T., 2008. "Overheads Reduction of the Distributed Time Warp Simulation". *In Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*. ACM, 26:IIIA.626:1.