Heuristic checkpointing on a parallel heterogeneous discrete event simulation system

Ricardo Parizotto[a], Braulio Adriano de Mello[a]

[a] *Universidade Federal da Fronteira Sul, Chapeco, SC, Brazil*

Abstract

The growth of the size and complexity of distributed simulations are increasing the general effort for coordinating their time synchronization. Especially, the hard overhead for managing the consistent states and checkpointing protocols impacts negatively on the general performance. Non coordinate strategies are able to reduce theses effects, however, they increase the risk of cascade rollbacks and the creation of useless checkpoints. Checkpointing management operations are costly in terms of time due to the waste on computation. This work introduces the specification and high level details of the implementation of a strategy to reduce the delay caused by useless checkpoints. Aiming to persist in the general performance of the simulation, our solution does not use any coordination messages. The proposed solution was implemented and validated in the DCB (Distributed Cosimulation Backbone) architecture, a heterogeneous and distributed simulation backbone. Results of the simulations using our strategy show that it created considerably fewer useless checkpoints without increasing significantly the rollback time.

Keywords:
Checkpoints, Discrete Event, Asynchronous, Prediction, Fault Tolerance

1. Introduction

New solutions for heterogeneous and distributed simulation systems have been proposed for supporting bigger and more complex models. Composability and distribution issues deal with a variety of techniques and methods to

Email addresses: ricardo.dparizotto@gmail.com (Ricardo Parizotto), bramlo@gmail.com (Braulio Adriano de Mello)

process events and components communication in order to execute the heaviest models. Currently, this trend has motivated the development of new strategies to manage the hybrid synchronization where optimist and conservative components cooperate. For instance, techniques to define the event time by dynamic model analysis as discussed in Kunz et al. (2012) and Fu et al. (2013) and Saker and Agbaria (2015)Wang et al. (2009).

In the distributed simulation, such as happens in the parallel simulation, the time constraints violation (LCC - Local Constraint Causality) is a central issue. The rollback solutions based on checkpoints to restore past consistent states, due to LCC violation, are largely studied when considering the asynchronous components. For instance, in Quaglia (1999) the authors introduced probabilistic checkpoints, in Sato et al. (2012) there was designed and modeled a non-blocking checkpointing system that extends a multi-level checkpointing system, in Braulio Adriano de Mello (2015) they describe an implementation of an uncoordinated checkpoints protocol to manage rollback operations while keeping the timing correctness of the components.

Even though the classic techniques based on checkpoints are able to solve LCC violations, they could increase the computational overhead and so the general simulation cost. This happens, for instance, when coordinated techniques are used reducing the simulation performance. On the other hand, non-coordinated techniques or even partially coordinated techniques are fault susceptible. They can not guarantee the correctness of the generated checkpoints.

This work presents an alternative strategy to avoid the generation of useless checkpoints based on non-coordinated techniques. Our strategy does not require exclusive management messages and it works monitoring the simulation messages in order to decide if each current simulation state is consistent and if a new checkpoint on this state has some chance to be useless. This decision is probabilistic and it is dynamically generated, for each checkpoint, based on heuristic analysis of the simulation behavior. We integrated the solution in the DCB (Distributed Simulation Backbone) architecture Braulio Adriano de Mello (2015). The results analysis of the comparative case studies show that our solution reduces the generation of useless checkpoints improving the simulation performance.

The DCB is a simulation architecture for the heterogeneous discrete simulation model. It offers resources for the cooperation of distributed and heterogeneous components preserving the originality and internal behavior of the components. The previous version of the DCB implements a technique to identify consistent states. However, there are no guarantees that the checkpoints will be useful before a rollback operation.

In the remainder of this paper, the Background section presents the main issues of rollback based on checkpoints followed by the main related works. The specification of the metrics, heuristics definition, implementation of functionalities into the DCB and results are presented in the next sections. Following this, the conclusions and the references are presented.

2. Background and motivation

Coordinate solutions has been preferentially adopted in studies about checkpointing management on distributed simulation systems. Such solutions are able to ensure the consistency of the states for generating recovery points, however, these solutions for coordinating the simulations are costly. This feature, in particular, has motivated the exploration of non-coordinated or semi-coordinated solutions for the implementation of rollback-based checkpointing techniques.

Rollback-based techniques act to store the system states which should be consistent at the checkpoints. Considering that time violations (or LCC failure) are inevitable in the parallel or distributed execution of asynchronous processes, the need is evident to reduce the delay generated by these types of failures. The delay is defined, basically, by the overhead generated by the actions set of the checkpoints themselves and by the amount of work that is lost after a failure settled between the last checkpoint and the failure time. Bouguerra et al. (2013)

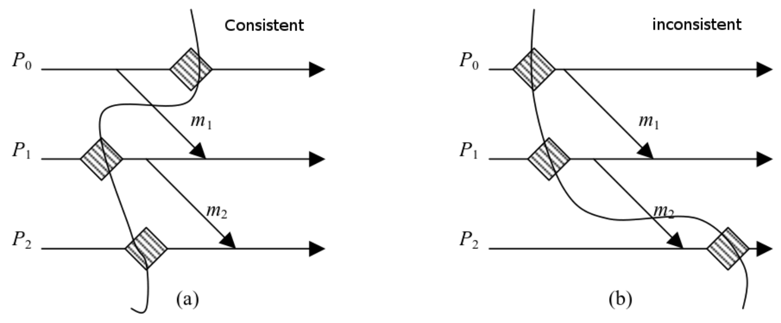

Figure 1: Consistent and inconsistent system states

An system state is consistent if there is no orphan message. A message is considered orphan if the receiver event was saved and its generator event was lost Kumar et al. (2010b). In figure 1 two global states are presented where 1(a) illustrates a consistent state and 1(b) illustrates an inconsistent event Elnozahy et al. (2002). The state 1(b) is inconsistent because the message M_2 is orphan. This message was received by the process P_2 but its generator event was not executed by P_1 yet.

The algorithms which take checkpoints with higher or lower frequency than needed could stimulate the creation of restart points that will no longer be in a consistent global state. Consequently they will never be used in rollback operations. Therefore, checkpoints that are saved on inconsistent states (useless checkpoints) are unwanted, since they do not contribute to the system recovery in a failure occurrence. Furthermore, they waste processing and storage resources.

Three different categories of rollback based checkpoints are presented in literature: Coordinated checkpoints, non coordinate checkpoints and communication induced checkpoints.

Coordinated protocols synchronize their checkpointing activities aiming to store only consistent global states. Coordination allows keeping only one checkpoint in stable memory, but it requires large latency involving coordination message exchanges.

Non-coordinated techniques allow each process to decide when to take checkpoints independently Braulio Adriano de Mello (2015). This means that every checkpoint is created without an intervention of any other processes. The main advantage of this is that each PL can take a checkpoint when it is most convenient without needing to exchange control messages. However, the processes are susceptible to cascading rollbacks.

Communication-induced checkpointing protocols use information obtained from simulation messages exchanged between processes to decide when to create checkpoints Saker and Agbaria (2015); Mattern (1989). This kind of protocol suffers less message overhead than coordinated protocols and they also can avoid the domino effect Reddy (2007).

2.1. Related Work

Some studies have been investigating strategies to combine the best features of different rollback techniques in order to find new solutions to manage the checkpointing with low overhead. This section introduces some of the main related works which are referenced in this paper.

Quaglia Quaglia (1999) proposed a checkpointing algorithm for optimistic simulations that combine periodic and probabilistic approaches. This algorithm decides if it should create a checkpoint or not before the execution of an event based on the probability of a rollback occurrence in a given time interval. However, as it remains working based on a constant time interval, useless checkpoints are not completely avoided.

Kumar et. al Kumar et al. (2010a) proposed a coordinated checkpointing algorithm in a mobile distributed system. It has a n sequential process which does not share the memory and the clock. The messages exchange is the only way of the processes communicating with each other.

The processes are not blocked for checkpoint generating purposes and only a subset of the processes are authorized to generate checkpoints. This algorithm presents a low cost in terms of processing, communicating on the wireless channels and memory consuming. Also, it avoids the checkpoints which are susceptible to the cascade rollback. However, as it does not estimate future rollbacks, the solution is not able to avoid the generation of unreachable checkpoints.

The processes coordination is based on a dependency vector. It works based on the message header. Consequently, it could increase strongly the message size when simulating hard complexity systems.

An uncoordinated checkpointing algorithm is presented in Braulio Adriano de Mello (2015). In this solution, each model component generates a checkpoint on a periodical time interval given by a finite number of events. The amount of these events is constant for each simulation. Although the algorithm ensures a low management overhead, it is able to generate useless checkpoints and it is susceptible to cascade rollback.

Saker and Agbaria Saker and Agbaria (2015) present a checkpointing protocol where the complexity of the communication depends on the application. In this solution, the processes store information about the others processes which performed some communicating operation since the last checkpoint generation. Then, it coordinates its events only with this processes group. As it is a coordinated protocol, the checkpoints are generated only on secure states. There are no functionalities to avoid the generation of unreachable checkpoints.

In this work we are focusing on a distributed and heterogeneous architecture of simulation where models with a high number of processes or components could result in an undesirable delay when coordinated strategies are used to manage checkpoints. The interest on this problem has instigated new studies to find better solutions in order to reduce the overhead of control messages. Then, this work presents a strategy to find the best interval between checkpoints based on the past behavior (events) of each component and without control messages exchanging as happens in the coordinated strategies. Our solution reduced the occurrences of useless checkpoints.

3. System Model

Asynchronous components may create checkpoints that are useless for reprocessing a simulation from a previous consistent state in any rollback. Reducing the number of times that useless checkpoints are created will eventually reduce the number of saved states and garbage collection operations without increasing rollback time. Therefore, in this study, we assume such as scenario an architecture that interconnects independent and heterogeneous components where communication and coordination operations are maintained by a subjacent layer of the system, and keeping these operations totally independent from the simulation model.

The processes of all components can be synchronous or asynchronous. The inner behavior of each process is independent of the communication and coordination operations that are established in the architecture of the subjacent layer. Therefore decisions about time advance, checkpointing and rollbacks are made by the architecture based on data obtained from communication ports between components and from the model configuration data, which define each destiny and source for messages exchanging.

The subjacent layer of our system stores a message log (M_{log}) for every logical process. The log keeps the received messages ordered by their timestamp while the simulation run. This means that the $n-th$ message stored in the log has the higher timestamp and we identify it as $M_{log}(n)$. The ideal size of the log is discussed in Johnson (1989).

The architecture of our system holds mechanisms to manage the communicating operations and maintaining the consistency of the model. For instance, the dependency vector, that is similar to that presented in et al. Johnson (1989), is supported by the subjacent instance, or simulation backbone, of every component in the simulation model. This vector manages an LVT approximation of all dependent processes. The LVT's approximation of processes can be equals to the timestamp of the most recently received message. The timestamp represents the virtual time when the message event should be processed. Definition 1 presents the method used in our strategy to identify the dependencies among processes that represent every component in the model.

Definition 1 (Lampord causality). *A process P_k depends on another process P_j (denoted by $P_k \Rightarrow P_j$) if P_j can send some message requesting the execution of some event in P_k during the simulation. If P_j depends on P_w (denoted by $P_j \Rightarrow P_w$), by transitivity, P_k also depends on the state of P_w (it means that $P_k \Rightarrow P_w$).*

Events, which can be executed by the processes, can be classified into three different types: sent events, received events and external events. When a process executes an event e that interferes with the state of another process, it must send a simulation message to that process. A simulation message is organized basically by the information about the event to be executed and about the virtual time (timestamp) when the event must be executed. When the destiny process receives the message, it performs another event e' in a way that e precedes e' (or $e < e'$) allowing a causality effect of e over e' Saker and Agbaria (2015); Lamport (1978)

Since the asynchronous processes are subjected to causality failures, checkpoints strategies are used to restore consistent states of the process, when such failures occur, ensuring the correct ordering of events. In this work, we assume that the initial state (LVT = 0) of each process seems as the first checkpoint. However, synchronous processes are not subjected to causality failures and they are not allowed to perform rollback operations. Then, behavior prediction strategies are adopted to determine the time barriers to the synchronous process (they ensure that no event occurs out of order Ferscha and Tripathi (1998)Fujimoto (2000)).

3.1. Useless checkpoints

Analyzing the conditions and variability of the system global state, when it is subjected to failures, we identified and characterized types of checkpoints that are useless. They were classified into three different categories: inconsistent, non-sufficient and unreachable checkpoints. These categories are illustrated by the scenario exemplified in figure 2.

In Figure 2 the process PL_0 communicates with PL_1 by sending a message whose timestamp is less than the LVT of PL_1. This scenario illustrates an LCC failure. So, the PL_1 must be rollbacked to the state C1, which is the last checkpoint saved before the timestamp of the message that caused the LCC

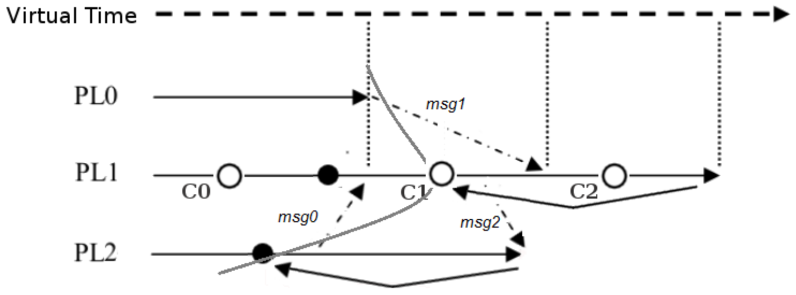

Figure 2: System state with useless checkpoints

failure. The LCC violation makes the message msg2 orphan. This situation requires a rollback also on the process PL_2 because that process executed other events after the event request received by messaging from the process PL_0.

The rollback operation executed by PL_2 generates another orphan message (msg0) and making the global state inconsistent (indicated in grey in the figure). Another rollback of the process PL_1 would eliminate the orphan messages and turn the global state consistent again.

In the scenario illustrated in Figure 2, we identify the first category of useless checkpoints: the inconsistent. A checkpoint is said to be inconsistent when a rollback sends the process to an instant of time before the checkpoint. In Figure 2, the checkpoint C_2 became inconsistent when the process PL_1 rollbacks to the C_1 because PL_1 received the message $msg1$ with the timestamp lesser than the time of the C_2.

The second category of useless checkpoints is called non-sufficient. A checkpoint is said to be non-sufficient if it is restored by some rollback operation but it doesn't belong to a consistent state. Therefore, one more rollback may be needed to restore a consistent state. Systems that allow the creating of non-sufficient checkpoints are susceptible to the domino-effect. They are different from inconsistent checkpoints because they are restored at least one time by a rollback operation. Nevertheless, they don't return the simulation state to a consistent state. In figure 2, the checkpoint C_1 becomes non-sufficient because when it is restored by a rollback operation it leaves one orphan message. So, a new rollback is executed by PL_1 in order to restore a previous checkpoint from C_1. It seems clear that the C_1 is useless and it impacted on one more rollback operation than required to restore the consistent state of that process.

The last category of useless checkpoints is named unreachable. Unreachable checkpoints are never restored by any rollback operation. They are created and stored by the process instance, but any failure always will be solved by newer checkpoints. This kind of checkpoint can be created by coordinate or non-coordinated protocols and depend on garbage collection techniques to be eliminated. In Figure 2, the checkpoint C_0 is unreachable since there is another checkpoint saved by the same process that makes part of a global consistent state and has its LVT higher than C_0.

4. Probabilistic Checkpoints

The previous section presented the definition of the three categories of useless checkpoints according to the system state at the moment of failure. In this section, we discuss some strategies for reducing the cost of the useless checkpoint generation. We assumed that the overall cost of the checkpointing could be reduced by preventing useless checkpoints and, consequently, avoiding LCC violations, and garbage collection operations and processing resources spending.

Section 4.1 presents heuristics which could be calculated based on information extracted from the simulation messages during the execution of the simulation. We assumed that the processes do not share a single global time and that the message's timestamp is used to estimate the local time of the sender processes for specifying the heuristics based on the metrics. The heuristics are used to identify two distinct strategies to calculate the rollback probability.

The first strategy works to identify communication 'patterns' between dependent processes to estimate the simulation time ahead of when new messages could be received. Processes are said to be dependents when they are directly connected to each other by a communication link. The heuristics are then used to identify which of the messages, according to the set of estimated messages of the connected processes, are able to generate time violations.

The second one calculates the rollback probability applying, initially, the first strategy. It means that it uses the first strategy to estimate the amount of foreseen events to a given future time interval of the simulation. Then, these events are compared to the average of the number of events between the past rollbacks for estimating the rollback probability and for deciding if a new checkpoint must be generated.

Finally, we proposed a probabilistic rule for deciding if each new checkpoint could be generated in scenarios where the components work independently of each other and the components are autonomous for generating checkpoints. The probabilistic rule is based on the probabilistic rollback for reducing the useless checkpoints as discussed in the next sections.

4.1. Heuristics

In this section, we define a set of heuristics to determine the probability of a rollback to occur during the simulation. Each of these heuristics has his notation and proposal explained in the section. How the simulation data is collected to calculate each one of the heuristics and their importance to the identification of the useless checkpoints is also discussed in this section. We settled the following heuristics:

- $timestamp_i$: timestamp of *ith* message saved on the log
- LVT_i: Estimation of the local virtual time of the process P_i
- N_{events}: Number of executed events during the simulation (including rollbacked events)
- $N_{rollbacks}$: Number of rollbacks executed during the simulation.
- $E_{rollback}$: Number of events that were scheduled since the last rollback
- $d(P_i, P_k)$: Distance between the processes P_i and P_k
- $f(P_k)$: Average time between received messages from P_k
- $t(P_i, P_k)$: Estimation of P_k number of events that would cause LCC violations in P_i

The heuristics like *LVT* and *timestamp* are already part of the synchronization mechanisms, therefore they don't need any new procedure to be implemented on the target architecture of this work. The heuristics $N_{eventos}$, $N_{rollbacks}$ and $E_{rollback}$ presented in Preiss et al. (1992) work basically as event counters, consequently they can be updated in constant time. These heuristics have their value updated every time that the local process executes an event. Further heuristics are updated only when the system creates a checkpoint.

The heuristics, which monitors the temporal distance between two processes, are defined as the difference of the processes' LVTs. In a scenario where the LVT of the process P_i is higher than the LVT of the process P_k and it is true that $P_i \Rightarrow P_k$, then the heuristic 'distance' represents a virtual (or simulation) time interval where the P_i can receive messages from the process P_k, which could cause an LCC failure. The equation 1 represents the distance heuristic.

$$d(P_i, P_k) = |LVT_i - LVT_k| \tag{1}$$

The exact calculation of the temporal distance between two processes requires the coordination between the processes. However, aiming to avoid the overload of the coordination protocols, we opted to estimate the distance between two processes using the timestamp from the received messages. For instance, when P_i calculates the distance from another process P_k, we use the timestamp of the last received message from P_k as an estimation of the LVT of P_k.

The calculation of the average number of messages from P_k to P_i when P_i has LVT higher than P_k is executed by dividing the distance between this two processes by the average size of received messages interval from P_k to P_i. This calculation allows us to estimate the number of messages that would be scheduled by P_k and received by P_i with their timestamps in the interval defined by $(LVT_i - d(PL_i, PL_k), LVT_i)$.

$$t(P_i, P_k) = d(P_i, P_k)/f(P_k) \tag{2}$$

The mean interval between received messages from P_k can be calculated using the *timestamp* of the messages saved in the log. Equation 3 demonstrates how to calculate the average interval between messages of P_k, where $timestamp_n$ and $timestamp_0$ refers, respectively, to the timestamp of most recent and older messages sent from P_k to P_i and n represents the number of messages saved in the log. In this scenario, the size of the log can interfere with the accuracy of the heuristic calculation. However, changing the size of the log will not change its time complexity, which is constant.

In this work, we use a log with no more than five messages for each dependent process. The ideal size for the log, considering the trade-off between space to store it and the heuristic accuracy is considered as perspectives to continue this work.

$$f(P_k) = (Timestamp_n - Timestamp_0)/n - 1 \tag{3}$$

Once the frequency of which each process generates events that create messages can have distinct distribution, the simplicity of our heuristic can have its accuracy reduced in some scenarios. Thus, the heuristic of the average interval may have a higher accuracy on a system where the interval between events that schedule messages are uniform. In models whose processes have different it behaviors could be necessary to adapt the frequency heuristic to another probability function. The identification of the most adequated probability function that a process uses to generate events, which could create messages, and its utilization to determine the simulation failures on simulation time also are seen as future works Kunz et al. (2012) Fu et al. (2013)Bouguerra et al. (2013).

4.2. Rollback Probability

This section discusses how to estimate the occurrence of rollbacks by using the heuristics presented in the previous section. We defined a new method for calculating the probability of rollback, which is executed whenever the simulation tries to generate a new checkpoint at a given time instant. The method obtains information from all the dependent processes to estimate if they are able to send messages that could create violations of LCC. This estimation is defined as a new heuristic called $p_{rollback}$.

The $p_{rollback}$ heuristic considers the events from all dependent processes and their respective chances to send messages which are able to create violations of LCC in the process that needs to generate a checkpoint. Then, we apply the heuristic presented in the previous section to each dependent process, according to equation 4 to estimate the arrival time of messages which may generate LCC violation. To do that, we do not include the processes whose LVT is bigger than the LVT of the process that is creating a checkpoint because they are not able to generate LCC violations.

$$P_e(P_i) = \sum_{c \in DP_i} t(P_i, c)[LVT_i > LVT_c] \tag{4}$$

The equation can be shortened as the sum of all processes P_k, which is dependent of P_i, where $LVT_k < LVT_i$, that could arrive at P_i in a virtual time interval higher than or equal to the time distance between these processes. Such arrived messages could generate LCC violations on P_i.

When at least one message whose timestamp is less than the local LVT is estimated, the probability of rollback is settled as 1. On the other hand, it is settled as 0 whenever there are no such messages estimated. There is no reason to identify the processes that will generate this rollback since any message with a timestamp less than LVT will cause LCC failure. Therefore, we assume that if $P_e(P_i) \geq 1$ at least one message would arrive with *timestamp* lesser than LVT of P_i. So, we used $P_e(P_i)$ as one way for calculating the rollbacking probability, since if the predicted message arrives then the processes will need a rollback to a previous state, making the checkpoint saved in this time as useless.

It is usually possible to monitor the number of rollbacks executed on large and complex systems. This monitoring allows us to estimate the average number of events between rollbacks Saker and Agbaria (2015). Equation 5 represents the average number of events that were executed between rollbacks Aupy et al. (2012).

$$\sigma = N_{events}/N_{rollbacks} \tag{5}$$

In a rollback situation, the sum of the number of messages that could cause LCC ($P_e(P_i)$) and the number of events scheduled since the last rollback will probably be higher than σ.

$$P_e(P_i) + E_{rollback} \geq \sigma \tag{6}$$

We calculate the occurrence probability of a rollback by multiplying both sides of equation 6 by α^{-1}.

$$(P_e(P_i) + E_{rollback})\sigma^{-1} \geq 1 \tag{7}$$

The equation 7 presents how to calculate the probability of the actual state of P_i being useless. If the result is higher than '1', then it indicates that more events would be scheduled than the average number of events between the rollbacks, indicating that the system is susceptible to rollbacks to a time lesser than the current LVT.

We presented two different strategies to calculate the $p_{rollback}$ heuristic. Both are considered as optimistic heuristics since we assumed that each process has its LVT at least equal to its last message timestamp saved in the log. The user can decide what strategy will be used depending on the needs of the system and information availability.

The computational complexity of the $p_{rollback}$ heuristic is higher than the complexity of the heuristics presented in the preview section. The complexity is as high as the size of the dependency vector. It happens because the vector must be entirely considered by $P_e(P_i)$ when $p_{rollback}$ is calculated.

The worst case scenario happens when the size of the dependency vector is equal to the number of process less one, which corresponds to the current process Saker and Agbaria (2015). As the other parameters to calculate this heuristic can be calculated in constant time, this heuristic must be a linear time algorithm $\Theta(n)$, where n is the number of LPs on the simulation system.

In recent studies, the distribution function of events scheduling frequency is calculated during execution using the probability density function Kunz et al. (2012) Fu et al. (2013). Though the cost of it can be higher, it can provide better accuracy in a heterogeneous environment where distributions used to schedule events can vary according to the model configuration.

4.3. Probabilistic decision

This section presents the strategy called 'probabilistic decision' for reducing the number of useless checkpoints. This strategy was specified based on the heuristics, which are discussed in the previous section.

During the creation of a checkpoint, whenever there is a high probability that the process executes a rollback to a previous state, the new checkpoint will also have a high chance to be useless.If the rollback is confirmed, then the checkpoint becomes inconsistent or non-sufficient. Therefore the probabilistic decision is calculated from the heuristics and estimates the chance of a rollback occurring to a moment earlier to the checkpoint creation.

The following expression synthesizes the probabilistic rule:

```
1: if ($(\alpha = rand()) < (1 - P_{rollback})$) then
2:     takes a checkpoint before executing the next event
```

The probabilistic decision was initially presented in Quaglia (1999) and adopted in this work such as the following: a value α is distributed uniformly between 0 and 1 and a checkpoint is created before the execution of the next event only if $\alpha < (1 - P_{rollback})$. The main difference to the first work is how we calculate $P_{rollback}$.

4.4. Complexity Analysis

The probabilistic decision provides real-time checkpoint scheduling. Whenever the system tries to make a new checkpoint, it happens if and only if the system is in a useful state according to the specifications noted previously. To achieve this, we take a probabilistic decision every time the scheduler tries to take a new checkpoint.

As the probabilistic decision requires the rollback probability calculation whenever it is executed, then the cost of the checkpoint creation increases proportionally to the complexity of heuristic time $\Theta(n)$. It happens because the dependency vector must be entirely considered by $P_e(P_i)$ every time the $p_{rollback}$ is calculated. The worst case is seen when the size of the vector is equal to the number n of the processes.

In Bouguerra et al. (2013) the authors define the delay by the sum of all checkpoints costs and the total time of the rollbacks. They also showed that by finding checkpoint scheduling in a way that the delay is the minimum, considering any failure distribution, it means a $\mathcal{NP}$-complete problem. Moreover, they proposed an optimum polynomial time algorithm considering that the checkpoint cost is constant. Though, in practical solutions, this is not always true just for full non-coordinate algorithms. In the coordinate algorithms, the checkpoints cost can be measured, for instance, by the message exchange complexity.

The checkpointing cost is increased by $\Theta(n)$ when using our approach. Concerning the probabilistic rules and the coordinated messages, the problem of checkpoints scheduling is an open matter in the heterogeneous systems. New studies could be regarding learning strategies.

5. Implementation

This section presents the integration of the probabilistic decision in the current version of the DCB prototype. We use a simple heterogeneous model to create scenarios with multiple useless checkpoints. The scenarios use a model that represents the behavior of a message passing system, which is similar to the one presented in Braulio Adriano de Mello (2015) and it integrates synchronous and asynchronous processes. Each process has its instance of the DCB layers. These layers are responsible for managing, translating messages and deciding when to create checkpoints without compromising independence or heterogeneity of the processes.

Whenever the system identifies a consistent state, then the probabilistic rules are applied to decide if that state represents or not a useless checkpoint. If the proposed heuristics indicates that the state can be useful for restoring the system from an LCC failure, then the processes should create a checkpoint. The pseudo-code presented in algorithm 1 illustrates the behavior of the methods involved in the implementation of the proposed solution by this work on the DCB architecture.

Algorithm 1 Probabilistic Decision on DCB

```
1: while Simulation in Progress do
2:     wait until system trys to checkpoint
3:     $\alpha \leftarrow random \in [0, 1]$
4:     if $\alpha > P_{rollback}$ then
5:         take a checkpoint
```

This method was implemented in the layer of the DCB, which is responsible for managing the received messages from the other components because this layer stores the messages log and the dependency vector. This information is used to calculate the rollback probability every time the system tries to create a checkpoint.

5.1. Study Case

The following processes were created to perform the study case: P_0, P_1, P_2 (assynchronous); P_3 (notime); and P_4 (synchronous). Figure 3 presents how these processes are organized. The values on the edges represent the probability of if one process communicates to another in a 100ms interval, making probabilistic the occurrence of external events with discrete time intervals. Processes are independent and communicate with each other through simple text messages. The simulation time 0 is seen by the DCB as the first checkpoint and it is the minimum limit for rollbacking operations.

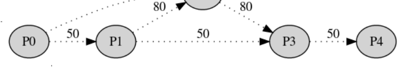

Figure 3: Message passing configuration graph

All the processes have the same internal behavior except for the way that they advance their virtual time. We configured scenarios for rollbacking occurrences to evaluate the probabilistic rules implemented in the DCB. Processes settled as synchronous increase their LVT in different ways when executing their events. The process P_0 increases its LVT in 90 units of virtual time after an event is scheduled, the process P_1 increases 95 and P_2 increases 100 units. Thereby we can observe some rollback occurrences, for instance, in the process P_2, caused by receiving late messages from P_0 e P_1.

To evaluate the heuristic checkpointing, we execute the same scenario using the checkpointing strategy of the previous version of the DCB and after we tested the solution presented in this work. Table 1 show how the checkpoints were created according to each one of the experiments.

Table 1: Properties of executions

Execution	Probabilistic Decision	Rollback Probability
I	-	-
II	v	$(P_e(P_i) + E_{rollback})\alpha^{-1}$
III	v	$P_e(P_i)$

The execution of the simulations was made on an Intel Core i5-3470 CPU @ 3.20GHz x 4 for $500,000$ units of virtual time. In the first execution, we used the previous version of the DCB where the time for creating checkpoints is estimated based on the identification of possible consistent states. In the executions II and III we activated the probabilistic rule to create checkpoints. However, the rollback probability was calculated based on different ways by the probabilistic rule: in II we applied the equation $P_{rollback} = (P_e(P_i) + E_{rollback})\alpha^{-1}$ and in III we applied the equation $P_{rollback} = P_e(P_i)$.

Figure 4 shows the number of the checkpoints which are created at the end of the each execution. We identified and counted those checkpoints that were useful by rollback operations and those that we did not use by rollback operations (unreachable or inconsistent).

The number of the useful checkpoint was far lesser in both executions in comparison with the strategy proposed by this work to create checkpoints. This result indicates that our strategy reduced the number of times that processes execute rollbacks. The number of useless checkpoints was also lesser in the executions with our strategies. The total of useless checkpoints means the sum of the inconsistent and the non-reachable checkpoints. Experiment II created 45,7% useless checkpoints less than experiment I and experiment III created 45,6% useless checkpoints less than experiment I. We did not use

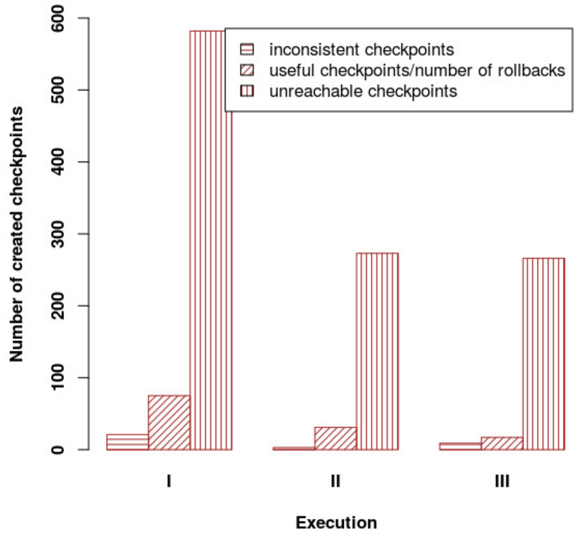

Figure 4: Proportion of checkpoint types created for each performed execution

the probabilistic rule in experiment I.

In Figure 5 we exhibit the total time of the rollbacks in each execution. The time of a *rollback* is equal to the difference between the instant of time of the failure occurrence and the time where the restored checkpoint was saved. The total time of the rollbacks is given by the sum of the time of all rollbacks executed during the simulation.

The total time of rollbacks executed in II and III was respectively 1.7% and 10.5% higher than rollbacks time obtained on I. The increase of the total time of the rollbacks can be explained by the fact that our strategy has not created some checkpoint that would have been useful. However, the total time of the rollbacks of experiments I, II and III were respectively 16.98%, 17.27% and 18.75% proportionally to the total simulated time. The size of the simulations was 500,000 units of time.

Analyzing both graphics we can observe that the number of useless checkpoints was considerably smaller when using the probabilistic rules, and without causing a significant increase of the total time of rollback. It means that the overall performance of the simulation had increased.

Metrics such as the number of created checkpoints and the total time

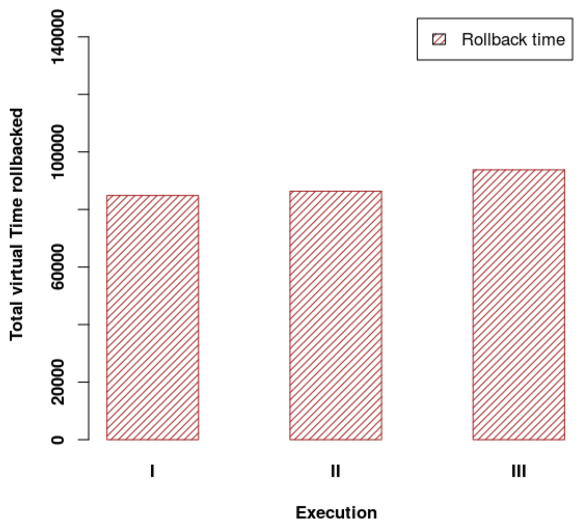

Figure 5: Total virtual time rollbacked for each performed execution

of rollbacks have had similar results at the end of both executions with our strategy. The fact that each of the executions was submitted to two different ways to calculate the probability of rollback indicates that they can provide similar results. In fact, this solution is able to predict future rollbacks without using any information about the behavior of the past rollbacks. This feature allows the system to avoid some useless checkpoints based on a rollback probability calculated without needing previous failures.

This work introduced an approach based on heuristics to avoiding useless checkpoints. Our solution does not depend on coordinated techniques and the heuristics are based on the information from the behavior of the simulation messages. Though this feature requires a small processing each time the simulation tries to create a checkpoint, it improves the general performance of the simulation by avoiding the overhead of management messages.

We defined a set of heuristics whose values are updated every time the processes execute an event. The probabilistic decision, initially proposed by Quaglia in Quaglia (1999), was extended in this work by applying the proposed heuristics. These heuristics support the probabilistic decision whenever the processes try to create a new checkpoint by estimating the time of the next LCC failure. It estimates the occurrence of LCC violation and it decides if a new checkpoint will be useful or useless on simulation time. The processes are not required to exchange coordination or synchronization messages, for instance, the null messages, to calculate our heuristics.

We implemented our strategy in the DCB, and we executed the scenarios for validation using a simple messages exchange model. The results showed a reduction of stored useless checkpoints in comparison with the previous versions of the DCB and established some trade-off in relation to rollback time. As our solution does not require synchronization messages, then the overall cost of the simulation tends to be reduced when compared with the solutions that depend on the control messages for coordinating the processes.

5.2. Future Work

We are investigating methods to figure out how to determine what will be the elapsed time of events dynamically and in advance. Then, it will be useful to improve the accuracy of the rollback probability regardless the behavior features of the model, for instance, when the faults are represented by an exponential or Weibull distribution.

How to estimate the best size of the log of the received messages and how to modify the structure of the messages to carry the rollbacking probability of the processes are being considered as perspectives. It means to include the result of the heuristic calculation of the sender process as a new attribute in the simulation messages. Since the synchronous components do not execute rollbacks, then these perspectives would make the simulation able to consider the distinction between synchronous and asynchronous processes for calculating the rollbacking probability. These methods could be useful to make the lookahead flexible on the synchronous components time management.

In addition, strategies for reducing the number of rollbacks are also able to reduce the overhead without negative impact on the checkpoint generation. As the contribution of this work had presented as an option for coordinated protocols, then a performing analysis between coordinated scenarios and our uncoordinated solution based on simulation heuristics is seen as another perspective. It could be helpful, for instance, for estimating the trade-off between the rollbacking time and the generation of the useless checkpoints.

Aupy, G., Robert, Y., Vivien, F., Zaidouni, D., 2012. Impact of fault prediction on checkpointing strategies. arXiv preprint arXiv:1207.6936.

Bouguerra, M.-S., Trystram, D., Wagner, F., 2013. Complexity analysis of checkpoint scheduling with variable costs. IEEE Transactions on Computers 62 (6), 1269–1275.

Braulio Adriano de Mello, F. M. M. C., 2015. Hybrid synchronization in the dcb based on uncoordinated checkpoints. Proceedings of ESM' 2015.

Elnozahy, E. N., Alvisi, L., Wang, Y.-M., Johnson, D. B., 2002. A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys (CSUR) 34 (3), 375–408.

Ferscha, A., Tripathi, S. K., 1998. Parallel and distributed simulation of discrete event systems. Tech. rep.

Fu, D., Becker, M., 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, p. 28.

Fujimoto, R. M., 2000. Parallel and distributed simulation systems. Vol. 300. Wiley New York.

Johnson, D. B., 1989. Distributed system fault tolerance using message logging and checkpointing. Tech. rep., DTIC Document.

Kumar, S., Chauhan, R., Kumar, P., 2010a. A low overhead minimum process global snapshot algorithm for mobile distributed systems. The International journal of Multimedia and Its Applications 2 (2).

Kumar, S., Chauhan, R., Kumar, P., 2010b. Real time snapshot collection algorithm for mobile distributed systems with minimum number of checkpoints. International Journal of Computer Applications 2 (9).

Kunz, G., Stoffers, M., Gross, J., Wehrle, K., 2012. Know thy simulation model: analyzing event interactions for probabilistic synchronization in parallel simulations. In: Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 119–128.

Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21 (7), 558–565.

Mattern, F., 1989. Virtual time and global states of distributed systems. Parallel and Distributed Algorithms 1 (23), 215–226.

Preiss, B. R., MacIntyre, I. D., Loucks, W. M., 1992. On the trade-off between time and space in optimistic parallel discrete-event simulation. In: Proceedings of. 1992 Workshop on Parallel and Distributed Simulation. Citeseer, pp. 33–42.

Quaglia, F., 1999. Combining periodic and probabilistic checkpointing in optimistic simulation. In: Parallel and Distributed Simulation, 1999. Proceedings. Thirteenth Workshop on. IEEE, pp. 109–116.

Reddy, S., 2007. Dynamic interval determination for pagelevel incremental checkpointing. Ph.D. thesis, National Institute of Technology Rourkela.

Saker, S., Agbaria, A., 2015. Communication pattern-based distributed snapshots in large-scale systems. In: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International. IEEE, pp. 1062–1071.

Sato, K., Maruyama, N., Mohror, K., Moody, A., Gamblin, T., de Supinski, B. R., 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, p. 19.

Wang, Y., Gao, S., Jia, Z., Li, X., 2009. Make a strategic decision using markov for dynamic checkpoint interval. In: IEEE Ninth International Conference on Computer and Information Technology. IEEE, pp. 197–202.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.