Closing the Gap Between Lookahead and Checkpointing to Provide Hybrid Synchronization

Edimar Roque Martello Junior[1], Acacia Terra[2], Ricardo Parizotto[3] Braulio Mello[1]

[1]Federal University of Fronteira Sul (UFFS)
Chapeco, SC – Brazil

[2]Federal University of Parana (UFPR)
Curitiba, PR – Brazil

[3]Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, RS – Brazil

{edimarjunior, braulio}@uffs.edu.br, acterra@inf.ufpr.br

rparizotto@inf.ufrgs.br

Abstract. *Hybrid synchronization provides more in-depth details about real distributed systems. However, several advances in algorithms to provide synchronization between local processes brings new difficulties to integrate into existing simulation architectures. This paper explores an alternative architecture to provide hybrid synchronization for decreasing resource wastings. We present optimistic and conservative synchronization primitives and design mechanisms to enable logical processes (LPs) to cooperate during the execution of a simulation. The results show that our primitives improve the simulation in terms of rollback-time and idleness.*

1. INTRODUCTION

Parallel Discrete Event Simulation (PDES) has been utilized as a technique to verify a broad range of complex systems. This has made a whole new field of research for computer science and several interdisciplinary areas. Employing distributed architectures to perform the simulation can scale the execution of specific scenarios. A distributed simulation is composed of logical processes (LPs) as units of execution that exchange timestamped messages for communicating. In runtime, synchronization primitives guarantee that event order is preserved, leading simulations to accurate results. Time synchronization can take many forms: (1) the utilization of lookahead to lock/free synchronous LPs; (2) the use of checkpoint/rollback techniques to restore asynchronous LPs to consistent states. In both cases, the Global Virtual Time (GVT) computation is an essential aspect of performance. It must be performed frequently enough to ensure simulation progress and free memory [Steinman et al. 1995].

Since Jefferson proposed time-warp and utilization of virtual time to synchronize distributed simulation systems, several optimizations for GVT computation have been proposed [Jefferson 1985, Jefferson 1990]. Mainly, the usage of time buckets to make lookahead flexible provided advances for conservative strategies. The utilization of failure prediction has shown efficiency in achieving faster results on opti-

mistic scenarios [Bouguerra et al. 2013]. Usually, the choice between these two techniques is made during modeling, and a hybrid strategy is not yet fully implemented [Jefferson and Barnes Jr 2017]. As such, creating several difficulties for modeling, once the developer cannot know in modeling time, which synchronization strategy would perform better in terms of CPU/Memory. This points to hybrid simulation strategies that support LPs running different synchronization primitives as the simulation executes [Eldabi et al. 2016] [Mikida and Kale 2019]. Such characteristics would keep simulation results accurate, saving resources in terms of checkpoints, rollbacks, and messages exchanged.

We present a hybrid synchronization architecture for distributed simulations. The architecture aims to provide different synchronization mechanisms that can interact during the simulation. Our main goal is to provide lookahead and checkpoint mechanisms and a coherent managing system to allow the cooperation between these mechanisms in runtime. Lookahead mechanisms are configured statically with periodic values. Dynamically, the architecture enables LPs to exchange control messages and change lookahead values according to the simulation's needs by each synchronous LP. At the same time, checkpoint algorithms for asynchronous LPs are configured with periodic values. Dynamically, each LP uses model-based heuristics to change the interval between checkpoints, aiming to reduce the number of rollbacks. Both algorithms have a reference for GVT values, and the algorithms for GVT calculation are adaptive.

The rest of this paper is organized as follows: Section 2 presents a brief background of synchronization for simulation and related work. Section 3 presents proposed hybrid synchronization in details. Section 4 describes implementation details, case studies and the experimental evaluation. Finally, Section 5 presents conclusions remarks and future work.

2. Background and Related Work

Several prior studies addressed strategies for improving methods and techniques at optimistic and conservative time synchronization on distributed simulation and integrating them into hybrid synchronization solutions. In general, the main challenge discussed by those works is how to preserve the order of the events without allowing time violation when exchanging event messages between optimistic and conservative LPs.

Simulation Architectures. Jefferson et al. [Jefferson and Barnes Jr 2017] proposed the Unified Virtual Time (UVT) architecture. Each LP can execute conservatively or optimistically, depending on the quality of the available lookahead information. The authors highlighted the dependency between the performance and the quality of the lookahead algorithms. However, the architecture focus only on parallel discrete event simulations. There are additional issues concerning distributed simulations to achieve the hybrid synchronization among distributed parts or LPs of a single model or even connecting distinct models [Fujimoto 2015]. Prior studies on distributed simulation have been mainly to focus on methods to reduce the distributed management cost by increasing the efficiency of rollback and lookahead operations. In this work, we introduce a distributed architecture that supports the hybrid synchronization among distributed and heterogeneous LPs. This architecture is based on two of our partial contributions, an optimistic synchronization method, and a dynamic estimation of lookahead.

Checkpoint Management. Reducing rollback costs while managing checkpoints efficiently is an open problem in the optimistic synchronization field. The Quaglia [Quaglia 1999] checkpointing algorithm works based on the probability of rollback occurrences in a specific time interval. The algorithm is able to decide if a new checkpoint should be created or not. However, useless checkpoints are not completely avoided. Saker and Agbaria [Saker and Agbaria 2015] proposed a checkpoint protocol that coordinates only with the LPs that it has communicated with since the last checkpoint. This solution reduces the coordination effort as well as the checkpointing frequency. However, local coordination still keeps some overhead when compared with non-coordinated solutions.

Lookahead Flexibilization. Besides, when considering hybrid synchronization, the limitations of the conservative LPs usually decrease the general performance of the simulation. Prior works proposed strategies to accelerate the performance of conservative LPs by introducing new algorithms to calculate the lookahead. Fu et. al. [Fu et al. 2013] presents the conditional lookahead (CLA) approach. Assuming that the lookahead between LPs is decreased temporarily on distributed discrete event simulations, the authors proposed a method to accelerate the simulation through the dynamic estimation of process-to-process lookahead. This approach can increase the lookahead ability by exchanging the CLA when it is stochastically too short.

In this work, we proposed a method to make the lookahead dynamic with low overhead. Each LP can calculate an individual lookahead based on the simulation message information. It allows getting closer to an ideal size of lookahead in a decentralized and flexible way. The next section presents the techniques, methods, and mechanisms that make up our hybrid synchronization architecture.

3. Hybrid Synchronization

In this section we describe the architecture for hybrid synchronization, which is general enough to include synchronous and asynchronous setups.

3.1. Overview

A simulation is composed as a set of *Logical Processes* (LPs), one per physical process. An LP behaves as a simulated logical process. All the interactions between physical processes are associated with a *timestamp*, which corresponds to a specific LP event. Ordering is required to preserve the causality of the simulation model. To achieve event ordering, all the conservatives LPs are synchronized with a *Global Virtual Time (GVT)*. Each LP also has a *Local Virtual Time (LVT)*, which is a reference for the actual simulation time the process is running. A hybrid synchronization architecture is responsible for two main tasks: providing barrier mechanisms for synchronous LPs and sate saving and restoration for asynchronous LPs. The GVT works as an upperbound barrier to the LVT for conservatives and a lower-bound for rollback operations. The architecture must be able to host the cooperation between synchronous and asynchronous processes. To this end, the architecture must have well-defined communication signals between each instance of the synchronization primitives.

3.2. Cooperation

The system performs several verifications to ensure the model does not create out-of-order operations. The verifications are made in two different ways: conservatively or optimisti-

cally. The conservative approach blocks the execution of synchronous components with the lookahead's aid to perform the check and avoid inconsistencies. The optimistic approach leverages checkpoint and rollback to restore the system after the model creates out-of-order events. Finally, a cooperation mechanism enables logical processes to run different primitives to preserve each simulation LP's main constraints.

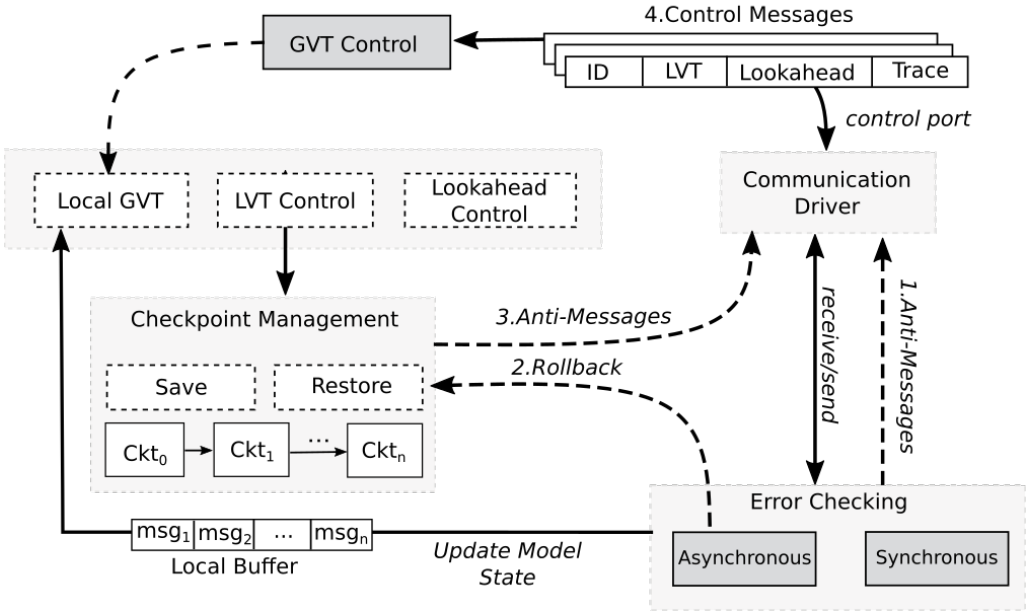


Figure 1. Hybrid synchronization overview

A completely hybrid behavior of the system depends on mechanisms to enable collaboration among synchronous and asynchronous processes. The challenge of this collaboration is defining the synchronous LPs actions when receiving messages from an asynchronous one, and vice versa. The architecture introduces an error-checking component while enabling the cooperation between LPs. The error-checking works transparently to the model, and verify inconsistencies before delivering a message to the receiver (*reactively*) and before sending error-prone messages (*pro-actively*):

- ***Reactive:*** The system discards violating messages, and an anti-message is sent to the origin (Figure 1, Step 1) to enable conservative processes to receive messages from optimistic LPs. The *anti-message* mechanism is used to "un-send" messages sent by the rolled-back events. The reasoning for this is as follows: once a conservative process cannot process earlier messages, we require the optimistic source to cancel it (e.g., by rolling back). Way say that the system performs verifications reactively: an LP that receives messages run verifications (conversely to performing it on the sender).
- ***Pro-Active:*** We need another mechanism to optimistic processes regarding computing messages from conservatives. As presented earlier, receiving a violating message requires the system to rollback (Figure 1, Step 2). In practice, a rollback operation loads messages received in a rollback interval and re-execute them in the correct order with the help of anti-messages. However, we decided that during a rollback, an LP sends anti-messages only for optimistic sources (Figure 1,

Step 3). We call such behavior a Pro-Active verification once it avoids errors before sending messages. Despite this may incur on synchronous LPs processing discarded messages, it ensures that ordering is preserved.

The architecture employs techniques to reduce the overhead of the hybrid synchronization. Next, we discuss these mechanisms in detail.

3.3. Wait-Free Synchronization

The system leverages checkpoint/rollback management mechanisms to deal with asynchronous LPs. Initially, the checkpoint configuration is made statically to define the time intervals to tentative checkpoints. Dynamically, each LP shares checkpoint IDs with dependent processes to identify secure states. Besides that, model-based heuristics [Elnozahy et al. 2002] are employed to decrease the number of useless checkpoints.

All the mechanisms have the following basic properties [Carvalho 2015]:

- Each LP stores the state locally.
- Checkpoints are managed by a process that runs in a background way concerning the LP.
- Each checkpoint saves all the LP values and the respective virtual time.
- The checkpoint before the failure time is restored when had occurred a time violation
- Messages between the checkpoint restored and the current time are canceled using anti-messages (Unless the source is synchronous).

Static checkpoints do not perform any coordination to create restart points. Conversely, when the system is configured with the static setup, each LP creates checkpoints within a static real-time interval. Then, we introduce an optimistic synchronization method based on the Quaglia algorithm that integrates our proposed architecture. In addition to the probability of rollback occurrence, our solution identifies communication patterns between dependent LPs for measuring the utility of a checkpoint at run-time and avoiding useless checkpoints. LPs are dependents when a sender LP sends a message to a receiver asking to execute an event [Lamport 1978]. Metrics such as the number of event messages, the average time interval between messages, the temporal distance between LPs, among others, are examples of communication patterns between dependent LPs.

Dynamic Interval. Before the simulation starts, the manager configures the initial time interval in which each LP will try to create checkpoints. The system creates checkpoints only at safety states, and the time interval between checkpoints is calculated dynamically. This strategy avoids an excessive number of saved states when compared to static strategies. The system employs two different strategies for identifying useless checkpoints. Firstly, the checkpoint ID's are sent into simulation messages to identify *zig-zag* paths in the simulation trace [Netzer and Xu 1995]. When an LP receives a message from an LP with a higher checkpoint ID, the receiver enters the unsafe state and cannot create checkpoints. Secondly, besides the coordination with checkpoint IDs, we rely on a prediction of the LP's LVT using the number of clock ticks that the LP made to reduce the number of useless checkpoints.

The checkpoint model tries to predict the communication patterns which can lead to time violation states [Parizotto and Mello 2019]. To this end, the interval of checkpoint

creation is adapted to use heuristics [Quaglia 1999]. The method uses the known past events from all dependent components to predict whether they will send new messages that create time violations at the components which are going to generate checkpoints. Next, we discuss the heuristics in detail.

Model Based Heuristics. In a scenario where the LVT of a process P_i is higher than the LVT of the process P_k, and it is true that P_i depends on P_k ($P_i \Rightarrow P_k$), the temporal distance, denoted as $d(i,k) = |LVT_i - LVT_k|$, represents a time interval in which received messages from P_k will be LCC (Local Causality Constraint) violation prone. When $P_i \Rightarrow P_k$, it means that P_k send message to P_i. Therefore, we use the distance as a weight for our heuristic. Let us denote the set of allowed events on the time interval $(LVT_i - d(i,k), LVT_i)$ as Γ. Computing Γ is impractical if the distance is large [Bouguerra et al. 2013]. Our method computes Γ efficiently based on a subset of received messages. It allows approximating the number of messages scheduled by P_k and received by P_i with $timestamp \in \Gamma$. Applying it to each dependent process of P_i (defined as DP_i), according to Algorithm 1, we have the number of arriving messages which may generate LCC violation on this interval.

Algorithm 1: Handling the checkpoint creation

```
   Data: DP_i, checkpoint[]
   Result: checkpoint[]
1  begin
2      //model based heuristics calculation
3      for k ∈ DP_i do
4          if LVT_i > LVT_k then
5              messages_i = messages_i + d(i,k)/f(k)
6      //probabilistic decision
7      if rand() > (1 − P_rollback) then
8          if ((messages_i + E_r) ≥ P_rollback) then
9              take a checkpoint
```

Whenever the system tries to create a checkpoint, the probabilistic choice is applied to determine if the current state represents a useless checkpoint (Algorithm 1, line 7). We extended the conditional clause with our method to measure the usability of a checkpoint, requiring that both the expressions are true to create a checkpoint [Fagin et al. 1994]. The measurement of $messages_i \geq 1$ (Algorithm 1, line 3-5) suggests that there is at least one estimated message whose *timestamp* is lesser than LVT_i. In the optimal scenario, $messages_i + E_r$ is the correct number of events in Γ. Therefore we compose $messages_i$ with the granularity of events since the last rollback to check if the measurement reaches the optimal scenario (Algorithm 1, line 8). If the proposed metrics indicate that the state can be useful for restoring the system from an LCC violation, the process creates a checkpoint. The reasoning reasoning behind that begins by assuming that a rollback is going to occur: if $messages_i + E_r \geq P_{rollback}$, it indicates that more events would be scheduled in Γ than the average number of events between rollbacks, therefore the system is rollback prone; Otherwise, if there are few events since the last rollback and few messages arriving in a way that $messages_i + E_r \geq P_{rollback}$ is not satisfied, by contradiction, it is probably not rollback prone and a checkpoint must not be created.

3.4. Lock/Wait Barriers

The architecture defines a centralized mechanism, **GVT Control**, that manages the value of GVT for all LPs. This mechanism combines the GVT value to the lookahead value on conservative LPs and exposes them to the LPs. Unfortunately, it is unknown whether the definition of static time was insufficient or overdone and could cause idleness, deadlocks, or even time violations between LPs.

Dynamic Computation. Lookahead is a safe time-frame, in which LPs will neither generate nor receive new events. In this work, instead of using Null messages [Fujimoto 2001], we use control messages that carry a more broad set of attributes [Cai and Turner 1995]. Each control message has header fields as an array that stores the message exchange route (trace) between LPs. Once obtained the trace information, and written in the respective field on control message, lookahead flexibility becomes possible. Control messages are sent to the output control port corresponding to the GVT controller (Figure 1, Step 4). Next, the GVT controller calculates a single value as the lookahead, based on the received messages, and add it to the GVT. Finally, the new GVT is sent to the conservative LPs. This centralized solution imposes a single lookahead to all conservative LPs. It may cause performance degradation. For overcoming this limitation, we decentralized this step by allowing each LP to calculate the lookahead locally. Next, we discuss the details of the lookahead decentralization.

Per-Process Lookahead. Dynamic lookahead based on the longest distance between processes with direct communication (through the route) reduces the number of time violations. However, since time violations may cause incorrect simulation results, they should be avoided. On the other hand, avoiding time violations may take too much time producing idleness of the simulation. In our approach, each LP can calculate its lookahead dynamically. The lookahead value of the LPs changes during the simulation according to the execution time of future scheduled messages. The goal is to get closer to an ideal size of lookahead and avoiding time violations or idleness. In our approach, the lookahead calculation is made locally. Each LP is responsible for achieving its value of lookahead, by making a simple calculus through the scheduled events.

The timestamp field of the control message means the scheduling time of the event to be executed [Cai and Turner 1995]. Then, each LP can advance to the next scheduled event according to the queued messages without generating time violations. The value of the timestamp field is obtained by the distance between the first queued timestamp and the LVT of the LP that received the control message. The rate of the control messages exchanged is reduced hugely by calculating lookahead locally. Also, the individual lookahead strategy has a significant decrease in the number of messages that generate time violations as well as LP downtime.

4. Experimental Evaluation

In this section, we address the evaluation details of the system architecture. The system is made over previous work, called DCB (Distributed Co-Simulation Backbone) [de Mello and Wagner 2002, Souza et al. 2003]. We adapted the original architecture of the system to support the design proposed in this paper and the new components to avoid time violations. Next, we present the specification of the study cases that we made to evaluate the system.

4.1. Case Studies

We created synthetic scenarios to simulate time violations and evaluate our approach's performance: (1) an asynchronous and (2) synchronous. Both models have five LPs with the same internal behavior, and that communicate with each other through exchanging simple messages in intervals between 100-200ms.

(a) Asynchronous **(b)** Synchronous

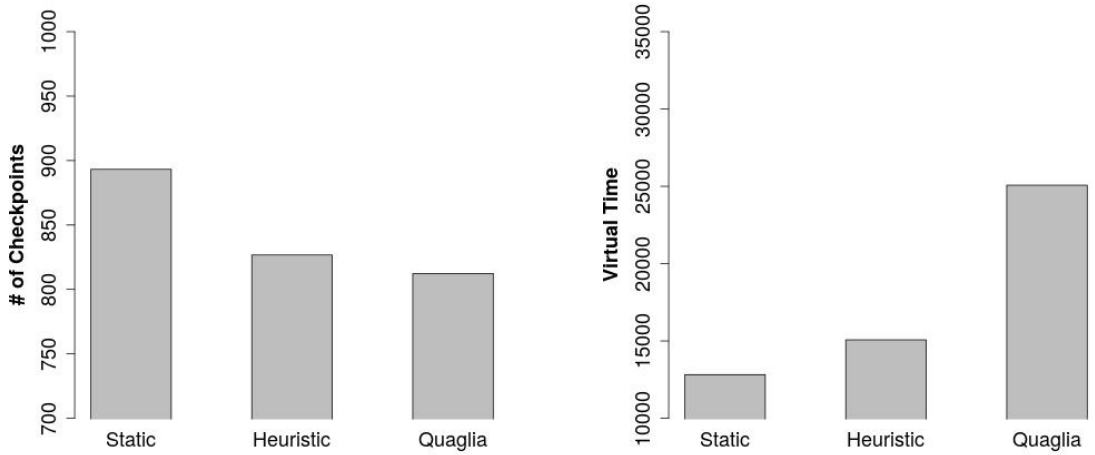
Figure 2. Configuration graphs of experiments use cases

Figure 2a depicts the communication graph for the asynchronous model. Each of the five vertices represents an LP, and the edges between the vertices represent the message exchange flow. Figure 2b describes the synchronous model. There is a cycle in the graph, which implies a more frequent message exchange between LPs, and consequently, the calculation of the lookahead is done more often. The next section discusses the evaluation in detail.

4.2. Results and analysis

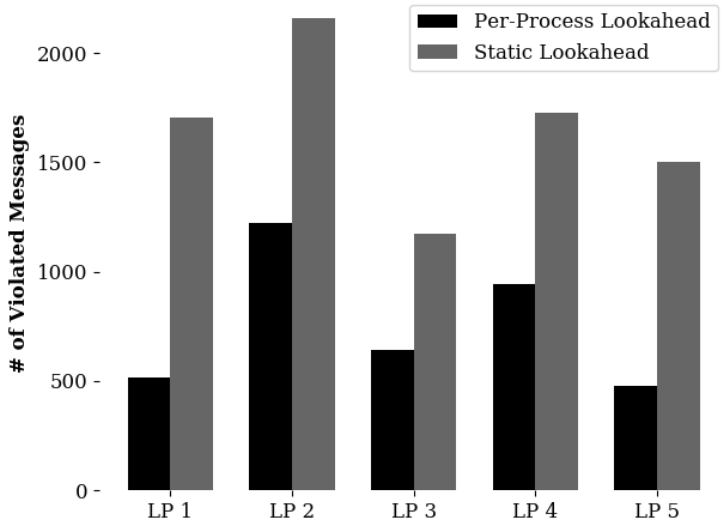
We executed the experiments with the models presented earlier to evaluate the performance of each synchronization mechanism. We performed ten simulations of one million units of virtual time for three different synchronization primitives. The experiments were performed on an Intel Core i5-3470 CPU @ 3.20GHz x 4 with 7,7 GiB of memory and running Antergos Linux 64-bit.

Checkpointing. To evaluate the impact of the checkpoint strategy is important to measure the number of checkpoints and the delay imposed by time spent on rollbacks. We configure the architecture in each case study for three different checkpoints methods: Static, where the system generates checkpoints according to deterministic and periodic intervals; the technique proposed by Quaglia [Quaglia 1999], and; the heuristic extension proposed in this work. Figure 3a presents the average of checkpoints. Results obtained with the Quaglia algorithm has shown the least number of checkpoints. The system created approximately 9% fewer checkpoints than with the static configuration and 2% fewer checkpoints than the heuristics proposed in this work. Figure 3b presents the average simulation time spent rollbacking. The time spent by one rollback is given by the difference between the time of a time violation and the checkpoint timestamp. The Quaglia algorithm spent more time performing rollback than static and heuristic approaches. The heuristic algorithm's time spent rollbacking was 17.6% higher than the algorithm with a static configuration, and the Quaglia algorithm was 95.5% higher than the static algorithm. It is explained the fact that the Quaglia algorithm did not create some checkpoints that would have been useful for reducing the time spent by rollbacking.



(a) Average number of checkpoints

(b) Average virtual time rollbacked



(c) Average number of violations

Figure 3. Simulation results for synchronous and asynchronous scenarios

Table 1 presents the average of rollbacks performed and the number of inconsistent and unreachable checkpoints, respectively. As an overall result, the heuristic method created less inconsistent checkpoints and consequently performed fewer rollbacks than both other strategies. The execution with the heuristic method generated 45% fewer rollbacks than the static and 15% than the Quaglia algorithm. The percents are similar to the inconsistent checkpoints. However, our algorithm generated 2% more unreachable ones than the Quaglia algorithm and spent around 35% less time executing rollback operations, which we argue is an acceptable tradeoff for a system like ours.

Lookahead. We configured two different lookahead approaches into the DCB environment: the (1) static and, (2) the per-process. In the static, lookahead receives a

	Rollback	Inconsistent	Unreachable
Static	11.72	6.4	875.06
Quaglia	7.64	3.8	800.96
Heuristic	6.48	2.92	817.32

Table 1. Statistics for each checkpoint method

fixed value (one hundred), which is added to GVT, so that the lookahead is applied to all processes present in the simulation. The per-process approach calculates lookahead locally and generates unique lookahead values for each LP. We executed each strategy five times. The collected metrics are the total simulation time, the total number of sent messages during the simulation, and the number of sent messages in past simulation times, i.e., messages generating time violations. In the per-process lookahead approach, the simulation ended with about 25% less total time (from thirty-eight minutes to twenty-eight minutes of simulation time). The number of sent messages was similar. Conversely, the number of time-violating messages has decreased. LPs 2, 3, and 4 presented a reduction of about 50% and the other LPs about 65%. The results can be seen in Figure 3c.

4.3. Discussions

Local changes to the synchronization mechanism create more difficulty in the management of the LPs. Changing the operation requires not only verification in both directions but the architecture to handle transient messages. We see as a perspective to allow automated changes between synchronization mechanisms by introducing a more in-depth check. To this end, we need ways to identify patterns between events and derive the moment in which it is necessary to swap between the synchronization mechanisms. Such improvements would improve the execution of the mode and make it easier for the developer to configure his scenarios [Bauer et al. 2005].

5. Conclusions

We have presented an architecture for hybrid synchronization on PDES distributed systems. The presented strategies combine efficient rollback and dynamic lookahead algorithms with a method to keep a reliable synchronization while exchanging timestamped messages among conservative and optimistic distributed LPs. Similar work, UVT [Jefferson and Barnes Jr 2017] supports hybrid synchronization; however, it focuses on parallel execution issues, and it does not integrate solutions to manage hybrid synchronization at distributed simulations. Although our solution achieves interoperability among distributed LPs, the system still has several limitations. We understand that the experimental evaluations are limited, and results cannot be generalized. We intend to evaluate the system with more realistic scenarios in the future. Finally, we see as an exciting future work to allow dynamic changes which affect the end-to-end behavior [Pellegrini and Quaglia 2015]. The deployment of part of the synchronization computation into the network infrastructure [Bosshart et al. 2014] and Integrating specification languages such as DEVS [Syriani et al. 2011] to define the behavior of LPs are also in perspective.

References

Bauer, D., Yaun, G., Carothers, C. D., Yuksel, M., and Kalyanaraman, S. (2005). Seven-o'clock: a new distributed gvt algorithm using network atomic operations. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 39–48. IEEE Computer Society.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

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

Cai, W. and Turner, S. (1995). An algorithm for reducing null-messages of cmb approach in parallel discrete event simulation. *Technical report*.

Carvalho, F. M. M., M. B. A. (2015). Hybrid synchronization in the dcb based on uncoordinated checkpoints. *Proceedings of ESM' 2015*.

de Mello, B. A. and Wagner, F. R. (2002). *A Standardized Co-simulation Backbone*, pages 181–192. Springer US, Boston, MA.

Eldabi, T., Balaban, M., Brailsford, S., Mustafee, N., Nance, R. E., Onggo, B. S., and Sargent, R. G. (2016). Hybrid simulation: Historical lessons, present challenges and futures. In *Proceedings of the 2016 Winter Simulation Conference*, WSC '16, pages 1388–1403, Piscataway, NJ, USA. IEEE Press.

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

Fagin, R., Fagin, R., Fagin, R., and Halpern, J. Y. (1994). Reasoning about knowledge and probability. *J. ACM*, 41(2):340–367.

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*, page 28. Society for Modeling & Simulation International.

Fujimoto, R. (2015). Parallel and distributed simulation. In *Proceedings of the 2015 Winter Simulation Conference*, WSC '15, pages 45–59, Piscataway, NJ, USA. IEEE Press.

Fujimoto, R. M. (2001). Parallel and distributed simulation systems. *Proceedings of the Winter Simulation Conference*, pages 147–157.

Jefferson, D. (1990). Virtual time ii: Storage management in conservative and optimistic systems. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, pages 75–89, New York, NY, USA. ACM.

Jefferson, D. R. (1985). Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3):404–425.

Jefferson, D. R. and Barnes Jr, P. D. (2017). Virtual time iii: Unification of conservative and optimistic synchronization in parallel discrete event simulation. In *Proceedings of the 2017 Winter Simulation Conference*, page 55. IEEE Press.

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

Mikida, E. and Kale, L. (2019). An adaptive non-blocking gvt algorithm. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '19, pages 25–36, New York, NY, USA. ACM.

Netzer, R. H. B. and Xu, J. (1995). Necessary and sufficient conditions for consistent global snapshots. *IEEE Trans. Parallel Distrib. Syst.*, 6(2):165–169.

Parizotto, R. and Mello, B. (2019). Uma abordagem para minimizar pontos de verificação inúteis em simulações otimistas distribuídas. In *Anais do XLVI Seminário Integrado de Software e Hardware*, pages 12–21, Porto Alegre, RS, Brasil. SBC.

Pellegrini, A. and Quaglia, F. (2015). Numa time warp. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '15, pages 59–70, New York, NY, USA. ACM.

Quaglia, F. (1999). Combining periodic and probabilistic checkpointing in optimistic simulation. In *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99.(Cat. No. PR00155)*, pages 109–116. IEEE.

Saker, S. and Agbaria, A. (2015). Communication pattern-based distributed snapshots in large-scale systems. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, IPDPSW '15, pages 1062–1071, Washington, DC, USA. IEEE Computer Society.

Souza, U. R. F., Sperb, J. K., de Mello, B. A., and Wagner, F. R. (2003). Tangram-virtual integration of heterogeneous ip components in a distributed co-simulation environment. In *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.*, pages 125–130. IEEE.

Steinman, J. S., Lee, C. A., Wilson, L. F., and Nicol, D. M. (1995). Global virtual time and distributed synchronization. In *Proceedings of the Ninth Workshop on Parallel and Distributed Simulation*, PADS '95, pages 139–148, Washington, DC, USA. IEEE Computer Society.

Syriani, E., Vangheluwe, H., and Al Mallah, A. (2011). Modelling and simulation-based design of a distributed devs simulator. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 3002–3016. IEEE.