Multiobjective Scheduling of Hybrid Synchronization Messages

Ricardo Parizotto[1] and Braulio Mello[2]

[1]Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, RS – Brazil

[2]Universidade Federal da Fronteira Sul (UFFS)
Chapeco, SC – Brazil

rparizotto@inf.ufrgs.br, braulio@uffs.edu.br

Abstract. *One of the essential aspects of distributed simulations is to order events according to a causal consistency model. However, traditional approaches are costly in terms of processing to ensure causality. A promising approach to order events is using a hybrid synchronization approach, where processes can alter dynamically between optimistic and conservative approaches. Unfortunately, synchronizing processes running a hybrid synchronization is a complex problem. In this work, we discuss a multi-objective scheduling of hybrid synchronization messages problem. Beyond that, we propose using a scheduling algorithm and describe how to integrate the algorithm in an existing distributed simulator. Finally, we propose a preliminary analysis of our algorithm regarding work done and the number of messages.*

1. Introdução

Synchronization is one of the fundamental aspects in exercising distributed simulations [Taylor 2019]. Traditionally, synchronization can be made optimistically or conservatively [Jefferson and Barnes 2017]. In its conservative (synchronous) manner, the simulation processes evolve in time using a time barrier defined by *lookahead* strategies. In the optimistic (asynchronous) version, the processes advance without any time barrier and use *global virtual time* (GVT) as a lower barrier to rollback operations. However, an entirely conservative approach can create idleness and reach deadlocks. On the other side, an optimistic approach can suffer from cascade rollbacks. A promising approach is to use a hybrid system, where processes that compose a distributed simulation can interoperate between optimistic and conservative mechanisms [Perumalla].

Recently, the hybrid synchronization paradigm motivated several different new approaches for synchronization. The UVT approach [Jefferson and Barnes 2017] makes processes change between conservative and optimistic methods dynamically during the simulation. Differently, Hybrid PDES [Eker et al. 2021] changes the entire simulation synchronization mode. Yet, another different approach enables conservative and optimistic processes to run at the same time during the simulation [Junior et al. 2020]. We focus mainly on the third approach, which is essential to provide properties such as composability. However, we argue that the abstractions here can be helpful for the other techniques because they can deliver out-of-order messages while migrating between different synchronization mechanisms.

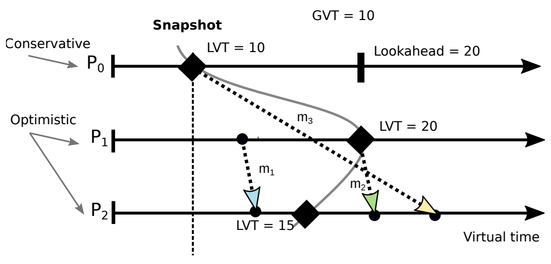
Unfortunately, synchronizing processes inter-operating between different synchronization approaches is a challenging problem. Specifically, the communication between conservative and optimistic processes can create events out of order, and imprecise results [Junior et al. 2020]. For example, a conservative process, receiving a message from an optimistic process, needs to discard messages that violate causality. On the other hand, if an optimistic process rolls back its state and discards messages from conservative processes, it compromises the simulation precision. We argue that it is not possible to achieve hybrid synchronization without compromising causal consistency. Still, new strategies are necessary to reduce the number of consistency violations and produce results with higher precision.

In this work, we investigate hybrid synchronization, focusing on reducing the number of consistency violations. We formulate the problem as a scheduling problem with multiple goals: (i) minimizing causality violations and (ii) maximizing the sum of work performed by the events processed. We devise an approach that intercepts messages in their receive and schedules them using a dynamic programming algorithm that outputs an equilibrium between these two goals. We also describe a vision of how distributed simulators could implement the algorithm proposed for both synchronous and asynchronous modes. Finally, we analyze our algorithm numerically regarding the number of messages scheduled and the work performed. Unlike existing work in this area that focuses on scheduling synchronous approaches, we focus on scheduling in hybrid scenarios. This brings new problems when synchronous and asynchronous processes communicate or when the simulation changes from one synchronization mode to another.

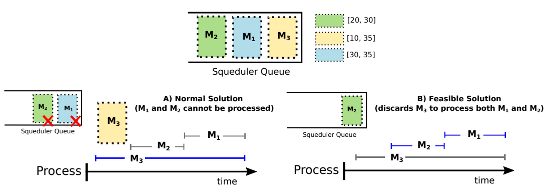
2. Motivation

We study the *Distributed Co-Simulation Backbone* (DCB) [Mello and Wagner 2002] as a representative example of a distributed simulator and present the challenges for synchronization in the context of the DCB architecture. These challenges are not limited to the DCB but are present in any distributed simulation system that uses hybrid synchronization. Specifically, we focus on how DCB manages the simulation messages and deliver them to processes. DCB keeps an input list of messages and uses the message $timestamp$ to establish the delivery. Each process has several attributes described by the user, defining how the process runs overtime. These attributes represent a temporal restriction that must be satisfied to messages do not violate causality. DCB process simulation messages using a scheduler that delivers messages when the process is on its initial time. To this end, the scheduler (1) selects between all the messages in the scheduler queue and ranks them based on its virtual time; (2) schedules the message for the virtual time of the process. However, this strategy can process a set of messages that is not the optimal according to the number of violations and the amount of computation performed.

Figure 1a exemplifies a scenario with three processes in a simulation. In the example P_0 is conservative and P_1 and P_2 are optimistic. Process P_2 is at LVT = 15 and already processed the message m_1, and it will soon process m_2. However, P_2 receives a message m_3 from the conservative process that has to be processed before the current time of P_2. Rolling the state back would enable the process to process the yellow message. However, processing m_3 implies leaving the two other messages without processing because of their beginning and end times (presented in more details in Figure 1b). We advocate that we can identify such situations during the message scheduling and



(a) An example of time diagram



(b) Message schedulling scenarios

choose the situation that provides more precise results and avoids unnecessary rollbacks.

Figure 1b show two message scheduling scenarios considering the example of Figure 1a. In scenario A), the standard solution schedules m_3 first. However, the system can not schedule other messages in the queue because of the message's virtual time limits. In scenario B), discarding message m_3 allows us to schedule both m_1 and m_2, which reduces the number of causality violations related to scenario A.

In this work, we propose the usage of a schedulling algorithm that is able to dynamically select a subset of messages that founds an equilibrium betwen the number of causality violations and the maximum of processing time.

3. Problem Definition

We consider as input a set of n messages $m_1, ...m_n$ with variable processing times. A message m_i must start its processing specifically in virtual time t_i and finish in d_i units of virtual time. Messages are processed by one single process, that composed a distributed simulation, an every process runs one message each time. We want to find a subset of messages that achieves the two following objectives.

- ***(1) Minimize the number of causality violations:*** In the conservative version, each causality violation means a message that the system can not process. In the optimistic version, each causality violation triggers rollbacks that consume simulation resources and make a message not to be processed. Thus this goal aims to minimize the number of causality violations and, as a consequence, maximize the number of messages processed.
- ***(2) Maximize the virtual time sum of complete messages:*** This goal is related to the need of choosing subsets of messages that contributes more to the simulation precision. Thus, the priority is given to subsets of messages that maximizes the sum of processed virtual time.

However, these two goals create a dichotomy: a subset of messages can satisfy the first goal, but another completely different subset of messages satisfies the second goal. Figure 2 present a plane messages-work. The green point represents a subset of messages that optimizes the number of messages being made. The blue point represents the subset of messages that would optimize only the work being made. Finally, we point (w,m) is a utopic point that would optimize both messages and work [1]. However, this point is not feasible in a scenario where messages conflict. In this work, we aim to provide ways to find the closest point to the one with the highest amount of messages and the highest amount of work made.

[1]in this work we use optimum point and utopic point interchangeably

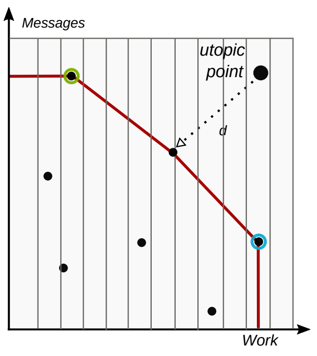


Figura 2. Scheduling plan with messages-work.

3.1. Synchronization constraints

In this work we are proposing a solution based on scheduling methods to deal with the dichotomy discussed above. Before presenting our proposed based scheduling solution, this section points some basic causality constraints of conservative and optimistic logical processes.

The GVT is used as lower or upper barrier depending on the synchronization approach of the logical processes on hybrid synchronization scenarios. Each logical process manages the evolution of its own time, called Local Virtual Time (LVT), according to the GVT. Conservative logical processes use the GVT and lookahead strategies to define the LVT upper barrier, and the timestamp of received messages must be greater than the GVT. Lookahead is a safe time-frame ahead the GVT, in which LPs will neither generate nor receive new events. Such requirements avoid causality violations between conservative processes. However, as we mentioned earlier, optimistic processes are not subject to these time constraints. Specifically, causality errors may occur in messages sent from conservative processes to optimistic ones.

Assuming that an optimistic process A sends a message m_t to a conservative process B, where t is the $timestamp$ of $A(m_t)$, the following condition must be satisfied to prevent causality violations on B:

$$LVT(B) \leq t \tag{1}$$

In this scenario, considering the use of the GVT to calculate $lookahead$, and also considering that the advance of $LVT(B)$ is limited to $GVT + lookahead$, for the sane scenario of processes A and B above, we assume that:

$$GVT + lookahead \leq t \tag{2}$$

and

$$LVT(B) \leq GVT + lookahead \tag{3}$$

As a result, there is no communication constraint between optimistic and conservative when $LVT(A) >= GVT + lookahead$. However, even using mechanisms such as promises implemented by *lookahead* strategies, rollbacks are still a problem and can break the assumption. Since these operations are not present in the behavior of conservative processes, the temporal constraints of conservative processes prevail over the optimistic. Thus, assuming that $A(m_t)$ is sent to B and

$$LVT(A) \leq GVT + lookahead \tag{4}$$

We consider that $A(m_t)$ may violate the causality of B, and thus, we consider this part of the problem of this work. The next section describes our proposed method for the previously mentioned dichotomy in the context of hybrid synchronization of distributed simulations.

4. Solving the Scheduling Problem

This section presents a method capable of finding a balance between minimizing the causality violations and the amount of work in the context of cooperation between optimistic and conservative.

Algorithm 1: Algorithm to schedule messages

```
Data: $utp$: the utopic point; $M$: the set of know and ordered messages; $V$:
      the set of visited messages
Function Receive($k$):
    $M \leftarrow M \cup \{k\}$
    mark all the messages higher than $k$ as not visited
    $m \leftarrow$ the last message of $M$
    Scheduling($m$)

Function Scheduling($m$):
    if $(m == 0)$ or $m \in V$ then
        return cost[m]
    select message n, such that n is the first message that does not conflicts
      with m
    $N[S] \leftarrow$ Scheduling($n$)
    select the next message, $k$
    $S[k] \leftarrow$ Scheduling($k$)
    return $min($Cost($S[k]$), Cost($m + S[n]$)$)$

Function Cost($k$):
    return $\sqrt{(k.work - utp.work)^2 + (k.msg - utp.msg)^2}$
```

The proposed method performs a scheduling analysis before delivering messages to simulation processes. We assume that we know what the set of non-processed messages is. Additionally, we also assume that we know received messages since the last consistent checkpoint of an optimistic process. To perform the scheduling analysis, we propose the usage of dynamic programming to find a subset of messages that reach an equilibrium between the objectives previously discussed. By running the algorithm, we identify the next message to be delivered to the simulation component.

Algorithm Idea. The main idea of the algorithm is to perform a search in a set of messages that selects a subset of messages that have the fewer distance to the utopic point. To start the search, we assume that messages are ordered in a non-decreasing order. The search starts by the last message an perform an recursive algorithm to reach the objectives mentioned earlier. The algorithm chooses between the two options: whether it is best to include the message of the corresponding recursion step with the option with having the last non-overlapping message (Figure 1, lines 10-11) or is best to consider the first overlapping message (Figure 1, lines 12-13). This choice is based on which of the options costs approximates more to the utopic point [Kolen et al. 2007]. To compute the cost, we move the set of messages into Cartesian coordinates into the euclidean space and calculate the euclidean distance between them: the utopic point to the point that represents the set of messages being analyzed (Figure 1, lines 17-18). We observe that the scheduling occurs after receiving a message k, using the costs of previous executions for messages with lower LVT than k. Thus after receiving a message, only the messages with the beginning time larger than the end time of the received message are marked as not visited (Figure 1, lines 10-11).

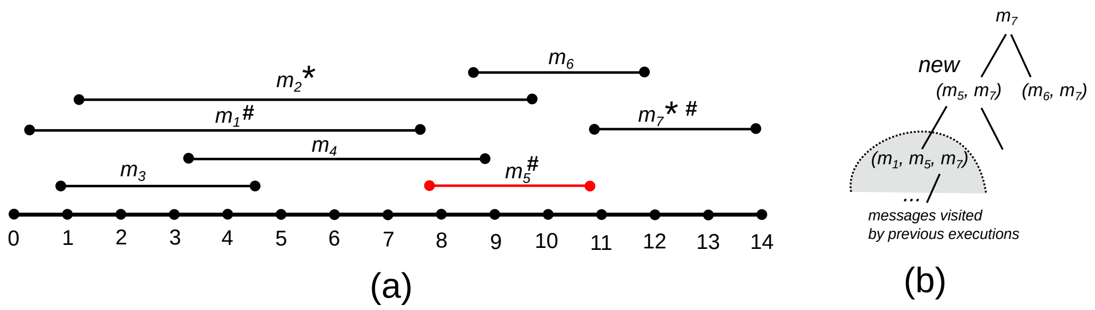


Figura 3. Illustration of the definitions of the algorithm 1

Figure 3 exemplifies a scenario where a new message, identified in red, arrives in the scheduler. In this scenario, every message before the received message was already visited by previous executions of the scheduling algorithm. In the previous execution, the solution A^* was optimal. After receiving the new message, creating a new scheduling plan is necessary to be closer to the optimum. In this case, the algorithm orders the new message with the previous ones and considers the subsequent messages the set of messages that still need to be explored. Conversely, the earlier messages in decreasing order were already visited by the previous execution and should not be revisited. The reasoning is that receiving the new message does not change the scheduling plan for these messages. Thus we can use the scheduling plan from the execution of previous messages in the receiving of the new one.

Correctness and Complexity. The correctness of the algorithm follows the sub-optimal structure of the problem. In each iteration, in order to select the messages we just take constant time. Since in every iteration the algorithm visits a message and does not visit it again in any other iteration, we only need N iterations to reach have the scheduling plan. The worst case scenario is when a received messaged have the lower LVT than the other messages. Since in this scenario each message needs to be visited at most once, this dynamic programming algorithm runs in linear time in the worst case scenario.

5. Preliminary Results

In this section, we present a numerical evaluation of our method. Our objective in the assessment is to understand how the proposed solution will behave concerning two metrics: (i) number of messages and (ii) sum of work performed.

We implement the proposed algorithm in Python and generate experiments that test the desired metrics to perform our evaluation. The experiments test the algorithm with a range of 10 to 100 entries in the scheduling queue, entries that relate to messages with a start and end time. We randomly determined the start and end times between 0 and 1000 time units. We run the presented scheduling algorithm 1000 times for each set of generated messages. For comparison purposes, we simulate the conservative execution of the same messages according to LTF policy. Below we describe the numerical results in more detail.

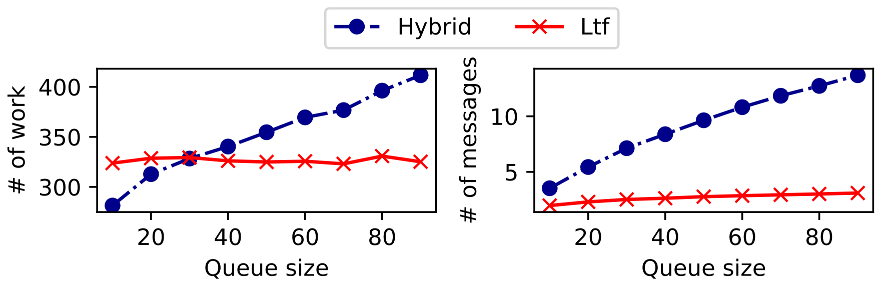


Figura 4. # of work and # of messages for each simulation

We first assess the amount of work done by each scheduling policy. Figure 4 shows the amount of work and messages scheduled by each approach, as well as the total for the set of messages tested. The amount of work performed using LTF remains almost constant as the number of messages increases. Using the approach proposed in this article, the amount of work performed increased as the number of messages in the input queue increased. We observe similar behavior for the number of messages executed, which means fewer causality violations would happen due to not processing some messages.

6. Related Work

Scheduling discrete event simulation events is not a new problem. This problem was explored in discrete simulation contexts that operate with only one synchronization mode (ex., only synchronous). Traditionally, a data structure is responsible for storing the set of events that have already been received but not yet executed. *Calendar Queues* [Brown 1988] is an example of a data structure used for this purpose.

From these structures, it is possible to use scheduling policies. For instance, in [Santoro and Quaglia 2010] the authors present a scheduler for optimistic simulation systems. The authors presented an implementation of a policy called *Lowest-Timestamp-First* (LTF) that can schedule events in constant time using a variation of Calendar Queues. Another example of a scheduling policy is *Probabilistic Scheduling* [Som and Sargent 1998]. This policy estimates the probability that an event to be processed will be lost in a future rollback operation and then schedules the event based on this estimate. However, both LFT and probabilistic policy only consider optimistic synchronization scenarios.

Recently, hybrid synchronization efforts have attracted community attention. In particular, *unified virtual time* (UVT) [Jefferson and Barnes 2017] is a conceptual architecture for hybrid synchronization able to dynamically switch from conservative to optimistic mode. In [Junior et al. 2020] an architecture for hybrid synchronization is presented and partially integrated into the DCB, where processes, differently from switching from conservative to optimistic mode, adapt their lookahead values and optimistic message cancellation techniques to avoid violations of time in the conservatives. More recently *Hybrid PDES* [Eker et al. 2021] presents a hybrid synchronization system that changes the synchronization mode of the entire simulation between conservative and optimistic modes according to a message distribution estimate.This work is complementary to the hybrid synchronization work since our goal is not to propose a new architecture. We propose a scheduling policy for hybrid synchronization.

7. Conclusions

In this work, we present the hybrid synchronization problem as a message scheduling problem. We propose a scheduling algorithm that seeks to find an equilibrium between the number of causality violations and the amount of work done. In addition, we specify how distributed simulators can use the method in distributed simulators. The preliminary results show the potential of the scheduling method compared the lowest timestamp first. In the future, we plan to implement this approach in a distributed simulator to test its feasibility.

Acknowledgements

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

Referências

Brown, R. (1988). Calendar queues: a fast 0 (1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227.

Eker, A., Arafa, Y., Badawy, A.-H. A., Santhi, N., Eidenbenz, S., and Ponomarev, D. (2021). Load-aware dynamic time synchronization in parallel discrete event simulation. pages 95–105.

Jefferson, D. R. and Barnes, P. D. (2017). Virtual time iii: unification of conservative and optimistic synchronization in parallel discrete event simulation. In *2017 Winter Simulation Conference (WSC)*, pages 786–797. IEEE.

Junior, E. M., Terra, A., Parizotto, R., and Mello, B. (2020). Closing the gap between lookahead and checkpointing to provide hybrid synchronization. In *Anais do XLVII Seminário Integrado de Software e Hardware*, pages 104–115, Porto Alegre, RS, Brasil. SBC.

Kolen, A. W., Lenstra, J. K., Papadimitriou, C. H., and Spieksma, F. C. (2007). Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543.

Mello, B. A. and Wagner, F. R. (2002). A standardized co-simulation backbone. In *SoC Design Methodologies*, pages 181–192. Springer.

Perumalla, K. S. /spl mu/sik-a micro-kernel for parallel/distributed simulation systems. In *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*, pages 59–68. IEEE.

Santoro, T. and Quaglia, F. (2010). A low-overhead constant-time ltf scheduler for optimistic simulation systems. In *The IEEE symposium on Computers and Communications*, pages 948–953. IEEE.

Som, T. K. and Sargent, R. G. (1998). A probabilistic event scheduling policy for optimistic parallel discrete event simulation. In *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation*, PADS '98, page 56–63, USA. IEEE Computer Society.

Taylor, S. J. (2019). Distributed simulation: state-of-the-art and potential for operational research. *European Journal of Operational Research*, 273(1):1–19.