Journal of the Brazilian Computer Society, 2024, 30:1, doi: 10.5753/jbcs.2024.3197
This work is licensed under a Creative Commons Attribution 4.0 International License.

Multiobjective message scheduling for Hybrid Synchronization in Distributed Simulations

Paulo Comasetto [**Federal University of Fronteira Sul** | ***paulogscomasetto@gmail.com***]
Ricardo Parizotto [**Federal University of Rio Grande do Sul** | ***rparizotto@inf.ufrgs.br***]
Braulio Mello [**Federal University of Fronteira Sul** | ***braulio@uffs.edu.br***]

Computer Science, Federal University of Fronteira Sul, Rodovia SC 484 - Km 02, Chapecó, SC, CEP 89815-899, Brazil.

Received: 28 February 2023 • **Accepted:** 12 November 2023 • **Published:** 05 July 2024

Abstract One of the essential aspects of distributed simulations is to order events according to a causal consistency model. Traditionally, implementing causal consistency can be made using a conservative or optimistic approach. However, traditional techniques are costly in processing time to ensure causality. A promising approach to order events is a hybrid synchronization approach, where processes can change 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 to reach an equilibrium between processing and causality violations and describe how to integrate the algorithm in an existing distributed simulator. The algorithm uses two memoization phases, making the scheduling suitable for a dynamic environment. Finally, to demonstrate the feasibility of our scheduling approach, we implemented it in an existing distributed simulation architecture. Analysis based on the experiments demonstrates the behavior of the simulation regarding the number of discarding/processed messages and work performed.

Keywords: Distributed Simulation, Hybrid synchronization, Multiobjetive scheduling

1 Introduction

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 a hybrid system, where processes composing a distributed simulation can interoperate between optimistic and conservative mechanisms Perumalla [2005].

Recently, the hybrid synchronization paradigm motivated several different new approaches for synchronization. The UVT approach Jefferson and Barnes [2017] makes processes change dynamically between conservative and optimistic methods during the simulation. Differently, Hybrid PDES Eker *et al*. [2021] changes the entire simulation synchronization mode. Yet, another approach enables conservative and optimistic processes to run simultaneously during the simulation Junior *et al*. [2020]. We focus mainly on the third approach, essential to provide properties such as composability. However, 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 interoperating 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, must discard messages that violate causality. However, discarding messages leads to another open problem on hybrid synchronization: Which messages can be discarded without compromising simulation results' accuracy or usefulness? On the other hand, if an optimistic process rolls back its state and discards messages from conservative processes, it compromises more than the accuracy but also the simulation performance. We argue that achieving hybrid synchronization without compromising causal consistency is impossible. Still, new strategies are necessary to reduce the number of consistency violations and produce results with higher precision.

In this work, we investigate the problem of scheduling messages in distributed simulations. We argue that it is possible to find the closest point between two objectives by message scheduling approaches: lowest causality violation and highest processed work. Specifically, we investigate simulations that can work using hybrid synchronization, focusing on reducing the number of consistency violations. We describe 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 reception and

schedules them using a dynamic programming algorithm that outputs an equilibrium between these two goals. Specifically, we leverage past executions of the scheduling and move the scheduling weights to the Cartesian plan to assign a scheduling plan.

Unlike our previous work Parizotto and Mello [2022], which only evaluated our algorithm numerically, in this extended version, we integrate the proposed algorithm into the DCB (Distributed Cosimulation Backbone) architecture Mello and Wagner [2002] to demonstrate the behavior of our solution under distributed simulation conditions. We ran synthetic simulation models in the simulator and compared the results with the traditional LTF scheduling policy. Finally, we analyzed the experiments' results regarding the number of messages scheduled, the number of discarded messages, the work performed, and the algorithm's runtime. Unlike existing work 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

The hybrid synchronization problem has been presented as a challenge in distributed simulation. Proposed approaches to synchronize optimistic and conservative logical processes do not solve the problem concerning the balance between avoidance causality violation and the sum of processed work. That is, how to define the set of discarding messages with the lowest impact on the time violation and precessed work?

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 and delivers the simulation messages 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 over time. These attributes represent a temporal restriction that must be satisfied so that messages do not violate causality. DCB processes simulation messages using a scheduler that delivers messages when the process is at 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 optimal according to the number of violations and the amount of computation performed.

Figure 1 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 P_2 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 detail in Figure 2).

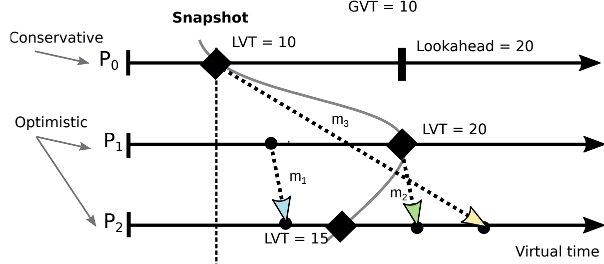

Figure 1. An example of time diagram

We advocate that we can identify such situations during the message scheduling. Then, we have proposed a scheduler that can choose the subset of the current messages, providing more precise results and avoiding unnecessary rollbacks. In addition, it avoids an increased number of discarded messages.

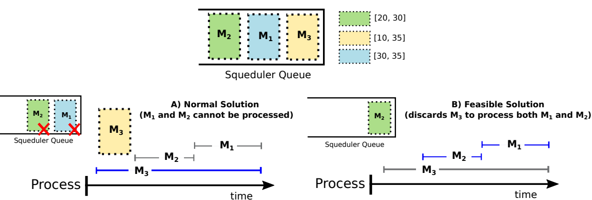

Figure 2. Message scheduling scenarios

Figure 2 shows two message scheduling scenarios considering the example of Figure 1. 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.

Such an example comparing two simple scenarios shows that the sum of work of the messages m_1 and m_2 is less than that of m_3. However, reducing the causality violations is an advantage during the simulation. Therefore, in this work, we propose using a scheduling algorithm to dynamically select a subset of messages that finds an equilibrium between the number of causality violations and the maximum 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, and 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 maximize the sum of processed virtual time.

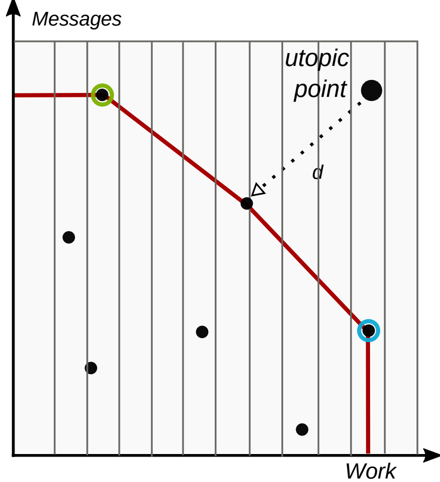

Figure 3. Scheduling plan with messages-work.

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 3 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.

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 of 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 same scenario of processes A and B above, we assume that:

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

and

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

As a result, there is no communication constraint between optimistic and conservative when $LVT(A) \geq GVT + lookahead$. 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, firstly, the main assumptions considered for scheduling messages in the input messages queue of the Logical Processes. Next, we present a method capable of finding a balance between minimizing the causality violations and the amount of work in cooperation between optimistic and conservative logical processes. Our proposed method allows for optimizing messages and work according to the problem definition discussed in section 3.

4.1 Assumptions and Overview

The scheduling method analyzes the messages' input queue before delivering messages to the simulation processes. We assume that the set C of non-processed messages, defined by $C = (m_1, m_2, ..., m_i)$, is known. We also assume that we know the received messages since the last consistent checkpoint of an optimistic process. In addition, all these messages are ordered in a non-decreasing order according to their timestamps. The starting virtual time (timestamp) is defined by t_i, and the amount of work is defined by d_i units of virtual time, as specified in section 3. Therefore, $m(t_i)$ gives

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

the timestamp of the message i, and $m(d_i)$ gives its work. By using dynamic programming, our proposed method finds a subset of messages that reach an equilibrium between causality time violations and the amount of work, as previously discussed. Then, given a set C of input queue messages, the algorithm identifies the next consistent message to be delivered to the Logical Process.

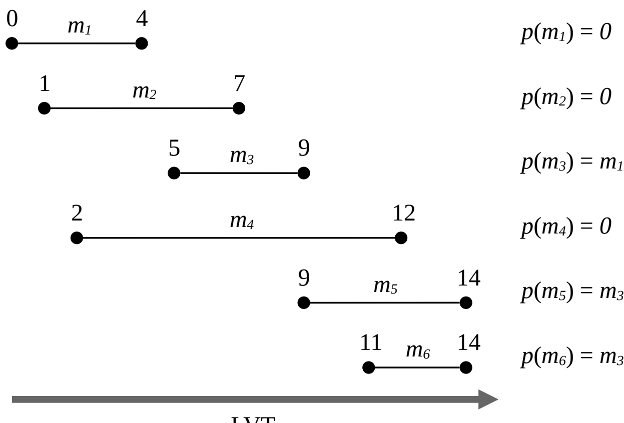

Figure 4. $p()$ function (adapted from: Kleinberg and Tardos [2006]).

Our solution is based on two main functions: the function *p* first identifies a subset of messages which are overlapped and outputs a subset of the next subsequent message. The function *utp* outputs a value for the best scheduling plan according to the highest amount of work and messages. These two functions are used in a scheduling algorithm inspired by the Kleinberg and Tardos [2006]). Our scheduling algorithm memorizes the computed weights (including the utp outputs for each message) during two phases: (1) the computation of a new scheduling plan and also (2) between multiple executions (i.e., between the arrivals of different messages). Next, we discuss the details of this algorithm.

4.2 Computing and Memorizing p.

The first of them is the $p()$ function Kleinberg and Tardos [2006]. Given a set C of queued messages, $p(m_i)$ identifies the closest lower-timestamped and non-overlapping message of m_i from C. Figure 4 shows a scenario with queued messages where: given the set $C = (m_1, m_2, m_3, m_4, m_5, m_6)$, then $p(m_5) = m_3$ is true because $m(t_5) \leq m(d_3)$, means a non-overlapping message. Further, $p(m_4)$ has no lower-timestamped with a non-overlapping message, then $p(m_4) = 0$.

Before scheduling and delivering a new message to an LP, we memoize the $p()$ of all messages in the input queue. This memoization strategy saves processing time since $p()$ does not always change after receiving a message. There are two scenarios where p() changes: (1) When the set of messages is static in the queue; or a new message has the largest timestamp between the others already enqueued. However, this is only sometimes valid in a distributed simulation environment. The arriving queue receives unordered messages and can discard messages due to causality violation. In addition, scheduling a message to a Logical Process can change another message's memorized values. Because of that, we need to define rules to change the memorized values

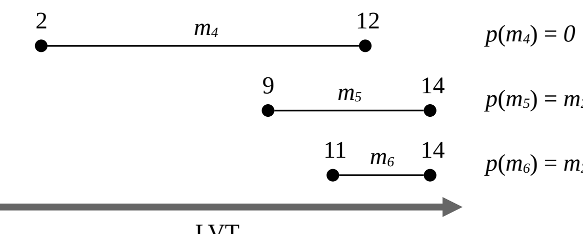

Figure 5. Memoization of $p()$ before m_3 arrives

in such situations. We argue that only messages with the timestamp of the new/delivered message must have their $p()$ calculated, justifying the memoization strategy. Therefore, we proposed the following memoization strategy.

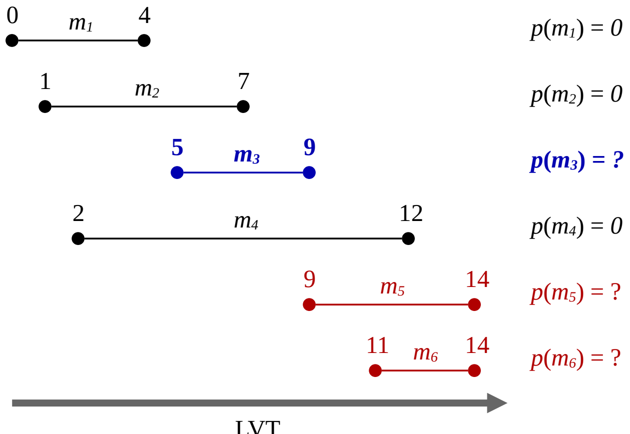

Figure 6. Memoization of $p()$ after arriving m_1 (adapted from: Kleinberg and Tardos [2006]).

Assuming:

Let $C = (m_1, m_2, ..., m_i)$ the set of queued messages

Let n the arriving or delivering message with $n(t_i)$ and $n(d_i)$;

Then:

for each $m \in C$ where $m(t_i) \geq n(d_i)$, $p(m)$ must be recalculated;

for each $m \in C$ where $m(t_i) < n(d_i)$, $p(m)$ remains valid.

To better understand how the memorization of p works, we illustrate two scenarios corresponding to the messages in the queue before and after a new message arrives. Figure 5 shows a scenario before m_3 arrives and the messages' respective $p()$ values. After m_3 arrives, as shown in Figure 6, we can see that the memorized values of $p()$ for the messages m_1, m_2, and m_4 do not change. Conversely, the $p()$ values of m_5 and m_6 are not valid anymore. They must be calculated again while calculating $p(m_3)$.

4.3 Computing and Memorizing *utp*.

The second function our scheduling algorithm uses is the utp introduced in our previous conference paper Parizotto and Mello [2022]. The utp function defines the notion of *utopic point*, and it is used as a parameter to find the sub-set of queued messages which represent the closest point to the highest amount of work and the highest amount of messages. Assuming a set C of input queued messages ordered by their increasing timestamp, and $m_i \in C$, then C_i is the sub-set of $m_{1,..,i}$ messages. And, $utp(m_i)$ defines the *utopic point* of m_i.

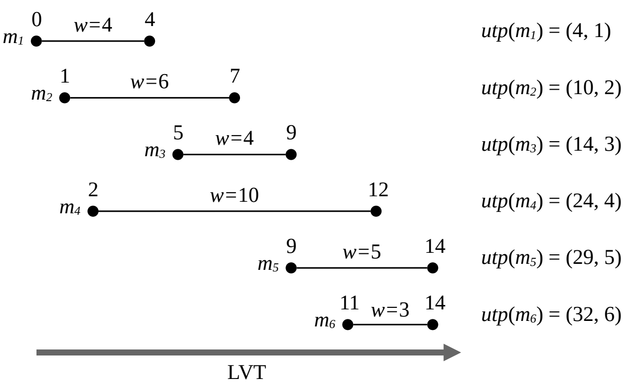

Figure 7. $utp()$ function

Figure 7 shows the same C as Figure 4. However, instead of the p, it shows each message's $utp()$. Assuming $w(m_i)$ the amount of work of the message m_i, then $utp(m_2)$ gives the sum of $w(m_1)$ and $w(m_2)$. The second information gives the number of messages that belong to the C_2.

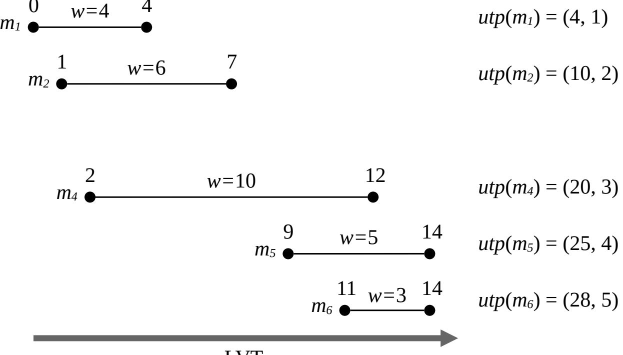

Figure 8. Memoization of $utp()$ function

Memoization procedures are also applied to the $utp()$ function. Whenever a m_j message is withdrawn from the set C_i, or a new message m_j arrives into the set C_i, then $utp(m_i)$ must be calculated again. If a new message m_j arrives and $j > i$, then $utp(m_i)$ remains valid.

Figure 8 illustrates the $utp()$ memoization before arriving m_3. Figure 9 shows the impact of the arriving m_3 message on the $utp()$ values of the set C_2 of messages. We can see that $utp(m_1)$ and $utp(m_2)$ do not change. The scheduler must execute the $utp()$ function for the new and existing messages where $j > 2$.

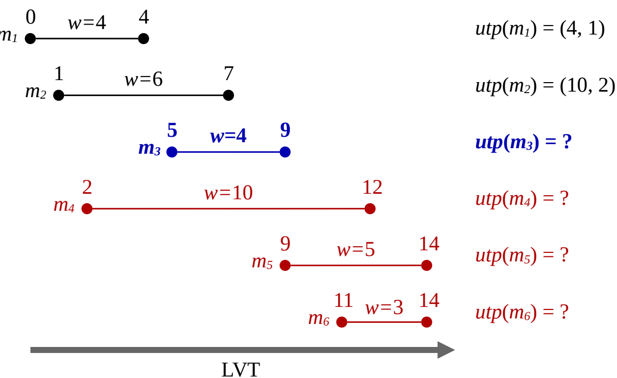

Figure 9. Memoization of $utp()$ function after arriving m_3

4.4 Scheduling Algorithm

After describing the essential functions we need to compute a new scheduling, we discuss how our scheduling algorithm employs those functions.

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

```
1  Function Receive(k):
2      M ← M ∪ {k}
       /* mark all the messages higher than k as
          not visited                                  */
3      V ← {a ∈ M|a < k}
4      Update MemoP
5      Update MemoUTP
6      m ← the last message of M
7      Memo[S] ← Scheduling(m)
8
9  Function Scheduling(m):
10     if (m == 0) or m ∈ V then
11         return memoS[m]
12     utp ← memoUTP[m]
       /* n is the first message that does not
          conflicts with m                             */
13     n ← memoP[m]
14     memoS[n] ← Scheduling(n)
15     S ← memoS[n] ∪ {m}
16     V ← V ∪ {m}
17     select the next message, k
18     memoS[k] ← Scheduling(k)
19     T ← memoS[k]
20     V ← V ∪ {k}
21     return min(Cost(S[k]), Cost(m + S[n]))
22
23 Function Cost(k):
24     return
         sqrt((k.work − utp.work)^2 + (k.msg − utp.msg)^2)
```

The algorithm performs a search in the set C_i of the m_j queued messages for selecting a subset of messages that have the fewer distance to the utopic point. The search performs a recursive algorithm to reach the sub-set of messages. The algorithm chooses between the two options: whether it is best to include the message of the corresponding recursion step

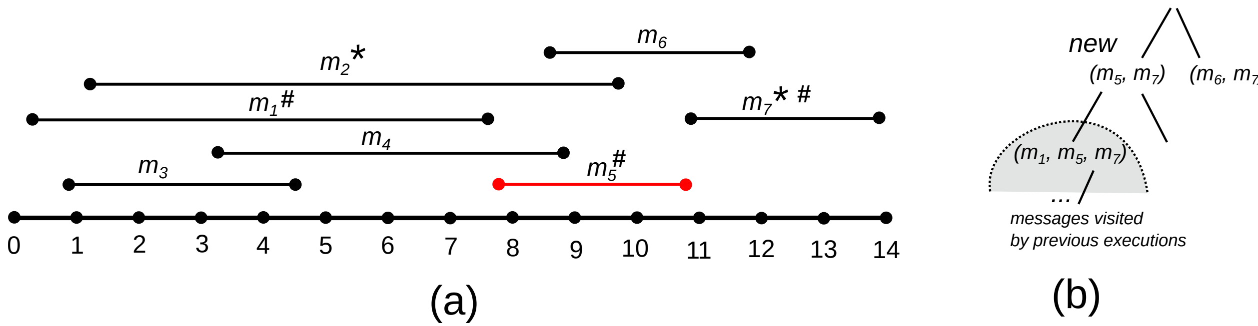

Figure 10. Illustration of the definitions of the algorithm 1

with the option with having the last non-overlapping message (Algorithm 1, lines 13-14) or is best to consider the first overlapping message (Algorithm 1, lines 17-18). 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 (Algorithm 1, lines 21-24). 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 a beginning time larger than the end time of the received message are marked as not visited (Algorithm 1, lines 2-7).

Figure 10 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 last 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 reuse the scheduling plan from the previous executions when receiving further messages.

Correctness and Complexity. The proof of the algorithm's correctness is based on the suboptimal structure of the problem. The proof is by induction and mirrors the one from the original weighted interval scheduling problem with a single objective.

Proof. Sketch: The base case with zero messages is trivially satisfied. Now consider that scheduling(k) is the optimal solution S_k. If k is *not in* the solution, the solution scheduling(k-1) is the same as scheduling(k) because k would conflict with the solution, including k-1. Otherwise, if k is *in* the schedulling solution S_k, this means that S_k is a subset of all the schedulling plans $S_0, S_1, ..., S_{i-1}$. Thus, the optimal solution is the minimum between those two.

Complexity analysis: The algorithm works by iterating through each message and selecting the optimal message to schedule based on its Local Virtual Time (LVT). We can analyze the complexity by looking at the following scenarios:

- Message insertion: once we receive a new message, we insert the message in a ordered list, and mark subsequent messages as not visited, which takes $O(n)$.
- Memoization and Assignments: In the scheduling part, the algorithm retrieves values from previous executions, selects a message in a list, and performs operations to memoize computed values, which only takes constant time $O(1)$.
- Recursive calls: The remainder of the algorithm performs recursive calls to visit earlier messages. Since the algorithm never revisits a message once it has been selected, the total number of operations necessary by each recursive call to the scheduling plan is $O(n)$.

The worst-case scenario occurs when a received message has the lowest LVT compared to all other messages. In this scenario, we must visit each message at least once. The best-case scenario occurs when the inserted message has the highest LVT in the list, requiring only $O(n)$.

Thus since the algorithm selects the optimal message in constant time for each iteration, the overall running time of the algorithm remains linear $O(n)$. We argue this makes the algorithm efficient and suitable for practical use in scheduling problems with large datasets.

5 Experiments and Result

In this section, we present the experiments and results. Our experiments aim to show the performance of a simulation in terms of the amount of work and number of causality violations under different simulation scenarios.

5.1 Workloads

We integrated our proposed solution into the existing Distributed Co-simulation Backbone (DCB) architecture. The previous version of DCB utilized the Least Timestamp First (LTF) scheduling policy, which was used as a point of comparison for our proposed solution. To evaluate the performance of our solution, we conducted experiments using nine different model configurations. These models were created by varying two key parameters: the total simulated time and the sent message rate. The rate of sent messages refers to the

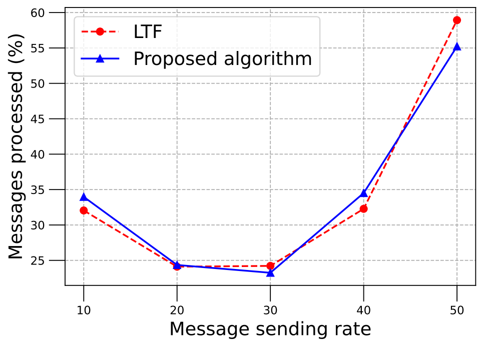

Figure 11. Messages processed: Increasing rate of sent messages

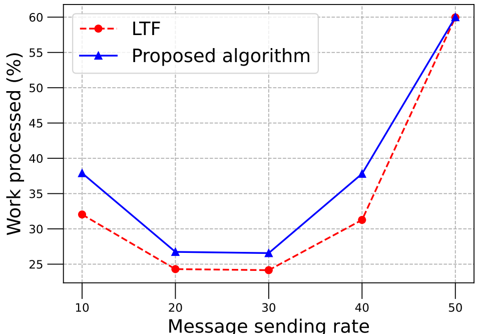

Figure 12. Work processed: Increasing rate of sent messages

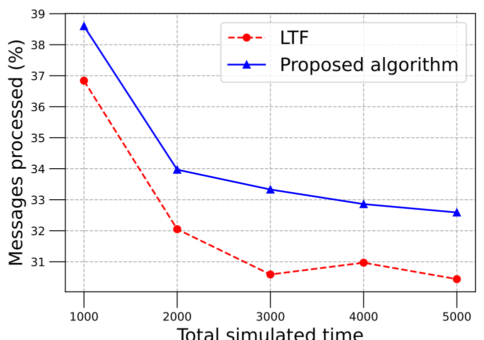

Figure 13. Messages processed: Increasing simulation size

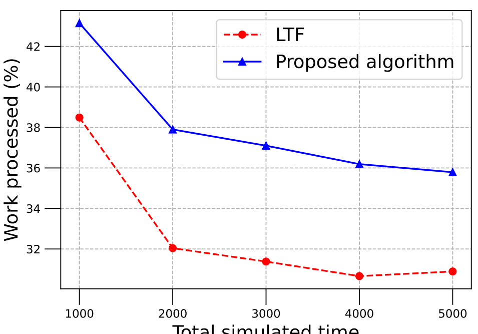

Figure 14. Work processed: Increasing simulation size

number of messages sent at each LVT in advance of the Logical Processes (LPs). To ensure that our results were statistically significant, we carefully designed our experiment configurations to consider the possible scenarios that may arise in a real-world implementation of the DCB architecture. This involved testing a range of values for the total simulated time and the sent message rate and analyzing the results to determine the optimal configuration for our solution.

Table 1 presents the nine configurations used in our experiments. The configurations can be divided into two groups. Models 1 to 5 have a fixed total simulated time, and the rate of sent messages increases linearly from 10 to 50. These models were designed to evaluate the impact of increasing the sent message rate on two important metrics: the number of causality violations and the amount of processed work. Models 1 and 6 to 9 have a fixed rate of sent messages, and the total simulated time increases linearly from 1000 to 5000. These models were designed to evaluate the impact of the simulation size on the results. The idea was to observe how the system behaves as the total simulated time increases and to determine if there is a relationship between the simulation size, the number of causality violations, and the amount of processed work.

We evaluated the impact of their proposed solution on the system by defining four key metrics: processed messages, discarded messages, processed work, and discarded work. These metrics allow us to evaluate the number of causality violations and the maximum processing time of the simulation events. The nine models were executed with the LTF scheduling policy and the proposed scheduling algorithm to compare

Model	Total simulated time	Rate of sent messagens
1	2000	10
2	2000	20
3	2000	30
4	2000	40
5	2000	50
6	1000	10
7	3000	10
8	4000	10
9	5000	10

Table 1. Models configuration

their effectiveness. The experiments were executed five times, and the arithmetic average of the replications was used for analysis to account for any variability and ensure representative results.

5.2 Changing the Rate

Experiments testing the impact of the increasing *rate* of sent messages scenario show that the proposed algorithm tends to reduce the performance at the highest *rates*. Figures 11 and 12 allow us to observe the results according to distinct *rates* based on *messages processed* and *work processed* metrics, respectively.

The proposed solution processes more messages with lower *rates*. However, the LTF scheduling policy demonstrated similar effectiveness on the highest *rates* (Figure 11). Similar results can be observed for the *work processed* metric. Figure 12 shows that both policies demonstrate related amounts of work processed on higher *rates*. Our solution

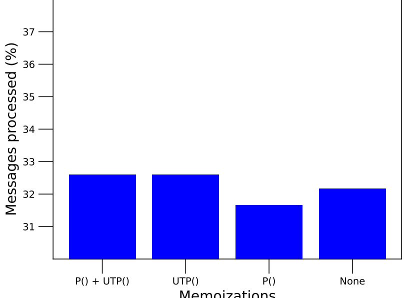

Figure 15. Messages processed

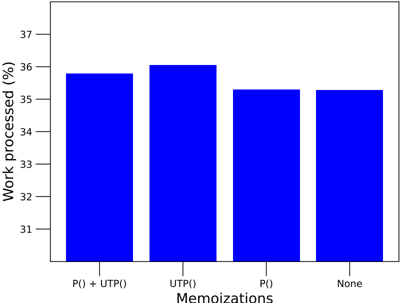

Figure 16. Work processed

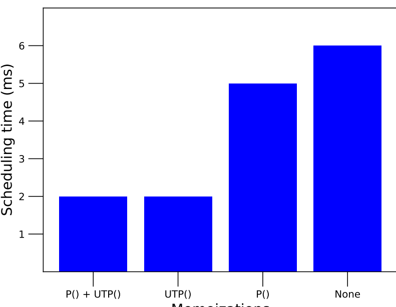

Figure 17. Scheduling time

presents advantages on lower *rates*. Increasing the number of messages in the input queue approximates the performance of both scheduling policies.

5.3 Impact of Increasing Simulation Size

In this section, we investigate the effect of increasing simulation size on system performance. To isolate the impact of simulation size, we keep the sent message rate fixed at 10. We present our results in Figures 13 and 14. These experiments provide insights into the scalability and help us understand how it performs as the size of the simulation grows.

Opposed to the results observed on changing the *rate*, the performance of the proposed algorithm demonstrated better effectiveness when the simulation size increased. Figure 13 shows that our solution presents a higher amount of *messages processed* than LTF on distinct simulation sizes. Increasing the simulation size does not affect the performance negatively. We can observe similar behavior of both algorithms on the graphic in Figure 14. Furthermore, we observe that the *work processed* percentage is higher than the *messages processed* percentage. Proportionally, our solution is more effective when considering the objective of maximizing the virtual time sum of *work processed*.

Our experimental results demonstrate that our proposed scheduling algorithm enhances message and work processing. The scalability and effectiveness of our approach become apparent through the experiments with increasing simulation size. Despite the challenges posed by hybrid synchronization in a simulation environment, our algorithm's performance outperforms LTF, as defined in the problem definition section. Our experiments generate many messages with overlapping execution times, leading to a low percentage of messages and work processed. Adopting a stress testing approach in the experiments enables us to evaluate the algorithm's effectiveness under worst-case workload conditions. In future work, we aim to assess the scheduling algorithm in a realistic simulation with hybrid synchronization. However, implementing such a simulation is an open problem that needs to be addressed.

5.4 Impact of Memoization on the Algorithm

To investigate the impact of memoization on the algorithm, we conducted a set of experiments with and without the memoization of $p()$ and $utp()$. We selected model 9 from Table 1 with a total simulated time of 5000 and a rate of sent messages of 10 for the experiment. To define the memoization variations, we considered the following scenarios: only memoizing $p()$, only memoizing $utp()$, and turning off both memoizations. Each variation was executed five times, and we used the arithmetic average of the replications for the analysis. Furthermore, we included the results of model 9 with both memoizations active from the previous experiments to provide a comparison baseline. We evaluated the performance of each variation in terms of message processing, work processing, and virtual time sum. The analysis of the experimental results revealed the impact of memoization on the algorithm's efficiency and effectiveness. The details of the experimental findings are discussed in this section.

As shown in Figures 15 and 16, removing the memoization of $utp()$ had a higher impact on the performance of the algorithm compared to removing the memoization of $p()$. Furthermore, the results indicate that the performance regarding messages processed and work processed reduced when both memoization techniques were disabled. This finding suggests that memoization is crucial for achieving better performance in the proposed algorithm. We argue that for larger simulations, this performance difference could be more significant and have a more substantial impact on the algorithm's overall performance.

Additionally, we investigated the impact of memoization on the temporal performance of the algorithm. New experiments were performed to measure the spent "scheduling time" in milliseconds. These experiments were conducted with and without the memoization of $p()$ and $utp()$ by using the same variations as before. The total simulated time was kept to 5000, as per model 9. However, we increased the model's rate of sent messages to 100 to overload the calculations of $p()$ and $utp()$ functions. Each variation was executed five times, and we used the arithmetic average of the replications for analysis purposes.

Figure 17 shows that when turning off both memoization yields, the average runtime is 6ms. When re-enabling them, the scheduling time drops to 2ms. We observed a relative reduction of $66.\overline{6}\%$. Also, we noticed that the memoization of only $utp()$ substantially reduces the runtime compared to the memoization of only $p()$. Despite the relative reduction obtained, the benefits of using both memoizations are low in absolute terms. We argue that the temporal difference could be intensified for more extensive simulations and simulations with more LPs, representing a more considerable impact on the algorithm's execution.

Table 2. Comparison with related work

Reference	Policies	Synchronization Technique
Santoro and Quaglia [2010]	LTF	Optimistic
Som and Sargent [1998]	Probabilistic Scheduling	Optimistic
Jefferson and Barnes [2017]	N/A	Hybrid
Junior *et al.* [2020]	N/A	Hybrid
Eker *et al.* [2021]	N/A	Hybrid
This work	Multi-objective Schedulling	Hybrid

5.5 Distributed experiments

Finally, we replicated the execution of all the models present in Table 1 in a distributed fashion. We used two machines for these experiments and reutilized the parameters, settings, and metrics from the parallel simulations already presented. Moreover, we sought to mitigate issues derived from the distribution of the models. For instance, network latency and messages in transit. This decision goal is to assess only the aforementioned metrics. Overall, the results obtained demonstrated that the distributed models had similar performance compared to their parallel counterparts. We reckon that, without alleviating the issues inherent to the distribution, there could be greater discrepancies in the outcome.

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 LTF 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.

Table 2 presents an overview of the scheduling policies and the respective synchronization algorithms used in related work. This work complements the hybrid synchronization approach since our goal is not to propose a new architecture. We present a scheduling policy for hybrid synchronization that requires handling more than one objective while choosing the messages it will process.

7 Conclusions

This work presents 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. We integrated the proposed scheduling algorithm into the DCB architecture. To evaluate its effectiveness, we conducted a series of experiments using a single simulation model with nine different configurations, varying the rate of sent messages and the simulation size. For each configuration, we ran five replications in the simulator. To compare the performance of our proposed scheduling policy against the traditional LTF scheduling policy, we executed the same set of experiments with both policies. We analyzed the number of messages scheduled, the number of discarded messages, and the work performed to assess the metrics of messages processed and work processed. Our results indicate that our scheduling policy outperforms LTF, particularly on larger simulation sizes.

Declarations

Acknowledgements

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

Competing interests

The authors declare that they have no competing interests.

References

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. DOI: 10.1145/63039.63045.

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. DOI: 0.1145/3437959.3459249.

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. DOI: 10.1109/WSC.2017.8247832.

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. DOI: 10.5753/semish.2020.11321.

Kleinberg, J. and Tardos, E. (2006). *Algorithm design*. Pearson Education India. Book.

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. DOI: 10.1002/nav.20231.

Mello, B. A. and Wagner, F. R. (2002). A standardized co-simulation backbone. In *SoC Design Methodologies*, pages 181–192. Springer. DOI: 10.1007/978-0-387-35597-$9_1$6.

Parizotto, R. and Mello, B. (2022). Multiobjective scheduling of hybrid synchronization messages. In *Anais do XLIX Seminário Integrado de Software e Hardware*, pages 49–57, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/semish.2022.222591.

Perumalla, K. S. (2005). /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. DOI: 10.1109/PADS.2005.1.

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. DOI: 10.1109/ISCC.2010.5546544.

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. Available at:https://dl.acm.org/doi/pdf/10.1145/278008.278016.

Taylor, S. J. (2019). Distributed simulation: state-of-the-art and potential for operational research. *European Journal of Operational Research*, 273(1):1–19. DOI: 10.1016/j.ejor.2018.04.032.