CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2016)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3811

SPECIAL ISSUE PAPER

Improving the performance of distributed discrete event simulation by exchange of conditional look-ahead

Desheng Fu, Matthias Becker*,† and Helena Szczerbicka

FG Simulation, Leibniz University of Hannover, Welfengarten 1, 30167 Hannover, Germany

SUMMARY

Distributed discrete event simulation is an important approach for enabling the modeling and analysis of the behavior of large systems. This approach presents a major problem, namely, the possible low performance due to the excessive overhead in synchronizing the distributed logical processes. To counter this, our approach to distributed discrete event simulation involves conservative synchronization and its acceleration using dynamic estimation of process-to-process look-ahead with a feedback mechanism. This mechanism allows for the estimation of a larger look-ahead, which may be invalidated and recalculated during the course of the simulation, if one of the processes obtains more detailed knowledge.

In this work, we extend the dynamically estimated look-ahead, on the basis of the local state of the logical processes, by exchanging conditional look-aheads, in conjunction with the broadcast of invalidation announcements. A notable reduction in runtime in various cases is thus achieved, especially when the estimated look-ahead is stochastically too conservative. Copyright © 2016 John Wiley & Sons, Ltd.

Received 18 January 2016; Accepted 28 January 2016

1. INTRODUCTION

The complexity of modeling and analysis of large systems can be handled using a *Distributed Discrete Event Simulation (DDES)* approach. Compared with sequential simulation, this has many advantages [1]. One specific goal is a reduction in execution time of the distributed simulation. However, this goal cannot be achieved in some cases. For example, if the *look-ahead (LA)* between tightly coupled *logical processes (LPs)* is very short [1].

One of the most important parameters in distributed simulation for describing the degree of coupling of LPs is the LA. A shorter LA means more causality errors in the parallel execution. There are two approaches for tackling this problem: avoidance of causality errors by *conservative synchronization* and resolution of the errors after the occurrence by *time warp* [1, 2]. Many conservative synchronization algorithms may be applicable, however, only when the LA is given for each link between LPs. A reduction in the performance of these algorithms is seen due to the *time-creep* problem, when the LA is too short. In this case, execution time of these simulations might be reduced by using a longer look-ahead.

Generally, a perfect prediction of the LA is impractical in the simulation of a complex model. This makes the estimation of LA necessary. This estimation has to be conservative in order to use conservative synchronization mechanisms, because any causality errors must be avoided here. There are also optimistic estimations of the LA to reduce the cost of time warp, but these are beyond the scope of this discussion. We shall only consider conservative estimations and allow for no *over-estimation*. In practice, a shortly estimated LA often does not imply that the LPs are tightly coupled,

*Correspondence to: Matthias Becker, Faculty for Electrical Engineering and Computer Science, Leibniz University of Hannover, Hannover, Germany
†E-mail: xmb@sim.uni-hannover.de

although the estimation algorithm should be improved nevertheless. One of the most important challenges of DDES is the estimation of LA, when conservative synchronization is applied to avoid causality errors.

We investigated the practical implementation of DDES with conservative synchronization and its performance enhancement through *dynamic estimation* of process-to-process LA in our previous work [3]. This dynamic estimation considers the run-time state of the model, in contrast to the *static estimation* of LA prior to the simulation, resulting in a better estimation and a larger LA. For example, the LA between two airports should increase, due to bad weather or traffic, which increases the duration of all flights. Such knowledge of the current run-time state of the model improves the dynamic estimation of LA during simulation and could provide a larger LA than static estimation. However, the overhead of the dynamic estimation at run-time is counted in the execution time of simulation. As a result, the estimation algorithm must be very efficient so that the overall execution time is reduced compared with static estimation. In other words, we should balance the quality and the overhead of the dynamic estimation.

Our approach aims at balancing the overhead of the dynamic estimation and the quality of the estimated LA. Our mechanism involves the exchange of *conditional look-ahead (CLA)* and broadcast of *invalidation announcement (IA)*. Additionally, we discuss some typical cases, where the approach is applicable. Our approach increases the LA between LPs in many cases, as shown in our evaluation. This approach has its greatest effect when the conservatively estimated LA is very short. However, the occurrence of an external events right after the expiration of the estimated LA is improbable. In other words, our approach provides the greatest performance boost, when the estimated LA might be too conservative from a stochastic viewpoint (in the simulation of multi-agent systems, stochastic Petri nets, Cell Biology [4], etc.).

The remainder of this article is organized as follows: the estimation of LA is introduced in 2. In Section 3, the relationship between LA and the time-creep problem is discussed. Our approach is then presented in Section 4. Section 5 shows a case study, as an evaluation of our approach. This is followed by Section 6.

2. RELATED WORK

Investigations of the dynamic estimation of the LA can be categorized into two groups. To prevent causality errors, the LA estimation for conservative synchronization algorithms (e.g., [5, 6]) has to be *conservative* in general. Such investigations are often called *look-ahead exploitation*. Conservative LA estimation is responsible for providing a larger LA for conservative synchronization algorithms, by estimating the run-time state. For example, *implicit look-ahead* in First Come First Served stochastic queuing networks was introduced by Nicol [7], which was extended by Lazowska to round robin [8] and some other queuing networks. The basic idea used by Nicol and Lazowska is the pre-simulation of predictable random behavior. During the analysis of stochastic queuing networks, the service times of processes are pre-sampled into a ‘future list’. The LA is then the measure of time to the end of next service, as per the future list. Another notable contribution was made by Meyer *et al.* [9]. In this investigation of conservative algorithms for wireless network simulation, the positive effect of dynamic LA estimation in wireless networks was reported. Liu and Nicol [10] also investigated the LA estimation in wireless networks with the simulator SWAN. They considered the IEEE 802.11 CSMA/CA distributed coordination function protocol on the media access control layer, and the LA was estimated from the CSMA/CA behavior of a station. The estimation of successor transmit time and potential push time on the medium access control layer leads to an increase in the LA and a significant decrease in execution time, especially in a low-traffic network. The other group of investigations estimate the LA for optimistic time-warp algorithms (e.g., [11]). In comparison with the estimation for conservative algorithms, *overestimation* is allowed. Such estimation failures will be repaired at a later stage, using the time-warp algorithm. These LA estimations are mainly responsible for providing a larger LA to reduce overhead in the algorithm. For example, Maritini *et al.* [12] introduced the *tolerant synchronization* and accepted event errors during the synchronization. The interval, during which event errors might occur, is predicted and the time-warp algorithm is applied to the time interval for the resolution of errors. Ferscha *et al.* [13, 14]

estimated the LA in Petri nets very optimistically in order to reduce the cost of the applied optimistic algorithm (time warp). This may also be considered as a conservative solution to the causality error without hard synchronization requirements. Consequently, the occurred causality errors are resolved by application of an optimistic algorithm. This idea is extended by Kunz *et al*. [15] as an improvement of the optimistic algorithm. Park *et al*. [16] introduced the *relaxed synchronization* to queuing networks. The LA is also estimated very optimistically for reduction in execution time.

In our previous work [3], we also presented a semi-conservative approach of LA estimation. This is an approximation, as overestimation and causality errors are allowed to a certain extent. By limiting the probability and the degree of overestimation, the level of approximation can be determined. The results show a possible reduction in execution time, while respecting the error bounds comparable with the exact simulation.

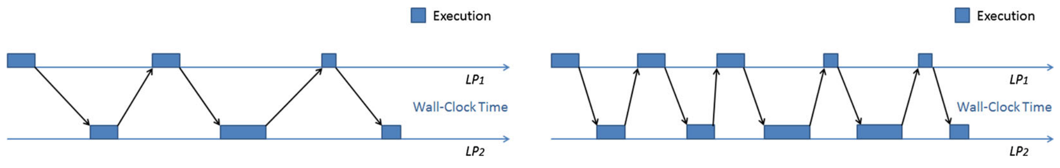
In our approach here, overestimation is allowed without the necessity of a time-warp algorithm, because causality errors are avoided. The concept of conditional LA makes this achievable by detection and invalidation of over-estimation [17], and the corrected estimation can be calculated conservatively. This will be discussed in the next section.

3. TIME-CREEP PROBLEM AND LOOK-AHEAD

As mentioned before, the well-known time-creep problem is one of the most important challenges in DDES with conservative synchronization. It occurs when the LA between tightly coupled LPs is estimated to be very short. To reduce the execution time of the simulation, sequential simulation could be taken into consideration instead of distributed simulation. When the process-to-process LAs are estimated dynamically, the distributed simulation should be preferred, when each process-to-process LA is large enough in general. However, we may have to deal with the temporary shortening of some LAs, or a small amount of LAs between certain LPs constantly being very short.

When the LA is very short, the LPs must synchronize with each other at a high frequency (time-creep). The costly synchronization must be thus performed very often. The time advance of the system will be very slow due to the extra cost. The advantage of distributed simulation will be eroded by the large overhead for synchronization between LPs. Most notably, the shortest LA in the whole model often determines the frequency of the synchronization.

The overhead of each synchronization is large for the following two reasons. Firstly, the synchronization, especially the global synchronization, reduces the parallelism, because the fast LPs must be blocked to wait for synchronization of the slow LPs. In addition, the cost of execution of the synchronization after the LPs are blocked is strongly dependant on the transmission delay between the LPs, including the time for processing received messages. In the idealistic situation, that is without any transmission delay, the performance is almost the same as with sequential simulation. The greater is the time needed for transmission, the slower is the advancement of time. Figure 1 shows an example, where the synchronization always takes place when an LP is blocked (such as the null-message algorithm [5, 6]). The effective time of execution with short transmission delay (b) is much longer than with long transmission delay (a). The transmission delay between the LPs is fairly different in practice. It is very short between two LPs, located at the same computer with shared memory, as compared with two LPs located at two different computers connected via LAN. The transmission delay further increases if the two computers are connected by the Internet.



(a) Time-Creep with long transmission delay

(b) Time-Creep with short transmission delay

Figure 1. Time creep for different transmission delays.

As discussed before, the overhead of synchronization might become large when the LA is short. To reduce overhead, we must distinguish between two situations: (1) The LA is intrinsically short. As mentioned earlier, the LA is a characteristic of the model and cannot be changed. The time creep problem in these situations is hard to avoid. Both LPs should be merged to improve the performance, or sequential simulation should be considered. (2) The LA is actually long enough, but the behavior of the model in the short future is hard to estimate conservatively. As a result, only a very short LA can be applied and the probability that an event will be executed in the short future after the expiration of LA is very small. A more precise (dynamic) estimation algorithm should be applied in such a situation. It is obvious that the dynamic estimation has its own cost, and this cost will be included to the total overhead of simulation; however, it can reduce the overall execution time in many cases.

4. OUR APPROACH

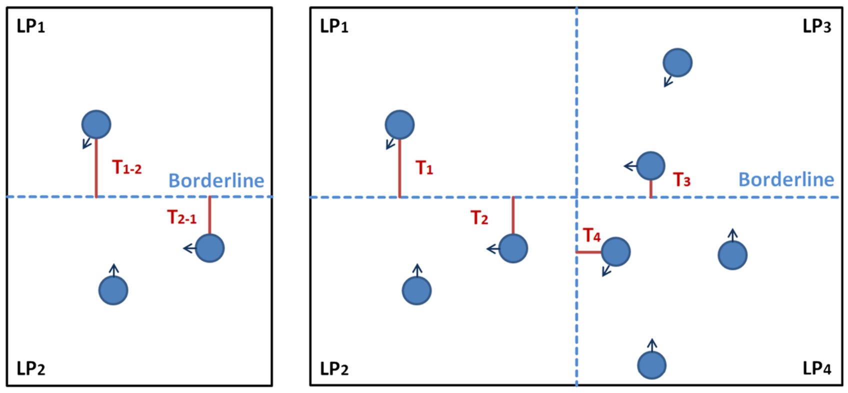
In this section, we present our approach to synchronize LPs with CLA and IA.

4.1. Solution with conditional look-ahead

Our approach for increasing the LA is realized with the introduction of the CLA. CLA is an extension of the original LA, but is only valid under certain conditions [10]. With the time advance, the conditions and CLA can be invalidated at any time, after the original LA expires. To ensure the conservative nature of CLA and to prevent causality errors, a method must be utilized to avoid the invalidation of CLA, until the related LP receives an IA. Moreover, it must be certain that the new LA, after applying the IA, is non-negative, so that no causality error occurs. The new LA can be sent with an IA. Alternatively, an IA may contain a timestamp. The IA will be 'executed' at this time with the information that the CLA is no longer valid and that the new LA after invalidation is zero.

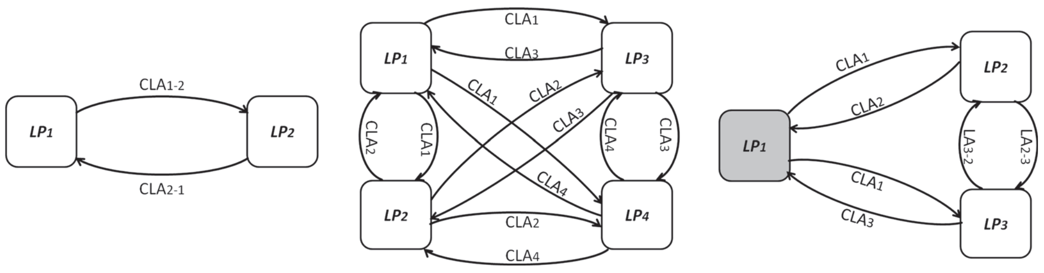
As mentioned earlier, the dynamic LA estimation during the simulation benefits from the knowledge of the current state of the model and therefore usually provides a larger LA than the static estimation. Two approaches exist for the state-dependent estimation. First, each LP estimates the LA based only on its local state (e.g., [5, 6]). The cost of the estimation is hence minimized, because unnecessary blocking for reading the state and exchange of the state between LPs is avoided. However, the estimated LA might be too conservative. Second, the global state is determined. Each LP or a central controller has to block all LPs to obtain the static global observation for the LA estimation (e.g., [1, 18, 19]). This approach will deliver a better LA, however, at the expense of overhead, because it blocks all LPs with a global synchronization each time the estimation is done. The implementation and performance obviously depend strongly on the model. It is a very interesting topic for investigation, but it is out of the scope of this article. Here, we want to present our solution based on CLA, which works almost independently of the model. Our solution combines two basic ideas. First, every LP exchanges certain information (CLA, IA) with *neighboring* LPs for providing a long LA. Second, no global synchronization is necessary.

A scenario is shown in Figure 2a as an example with a distributed multi-agent system simulation. In this example, the whole area is divided by a borderline, and each part is simulated with one LP (LP_1 and LP_2). There are several agents with built-in intelligence, which can move in the area to solve some certain problems. In general, simulation of such a DDES system is performed with time warp or a conservative synchronization with a global controller. No positive LA can be considered based on local state, because no information about agents in other parts of the area are given. One agent might cross over the borderline (hand-in) by now and immediately returns back (hand-out). As a result, we cannot utilize distributed conservative synchronization without central intelligence. The cost of time warp depends on the cost of duplicating the agents, along with the state of their intelligence, which can be very costly. For this reason, we may want to simulate the model with conservative synchronization. In this case, the conservative synchronization with central intelligence could be considered, for example, *synchronized execution algorithms* [1, 18] with a specified synchronization algorithm (e.g., [20]). This includes a global view of the whole area. Because no agent is right on the borderline at present, the exchange of information between LPs in



(a) Multi-agent system with two LPs (b) Multi-agent system with four LPs

Figure 2. Multi-agent systems.



(a) Model with two LPs (b) Model with more LPs (Fully connected) (c) Model with more LPs (Two groups)

Figure 3. Applicability of CLA in different topologies.

the short future is not necessary and the bi-directional LA could be calculated as the minimal time needed for an agent to move to the borderline regarding the position, as well as the maximal speed of each agent (T_{2-1} in the example). There exist various algorithms to synchronize the LPs with the estimated LA, which are beyond the scope of this article.

Without global observation, this idea could still be executed through exchange of extra information between LPs. This could be formally described with CLA and IA as shown in Figure 3a. The condition of the CLA sent from LP_1 to LP_2 (CLA_{1-2}) is that LP_2 will make no change to the local state of LP_1 (no external event or IA from LP_2 will be processed by LP_1). The length of CLA_{1-2} depends on the model. It is at least the LA, but it is larger than the LA in many cases (T_{1-2} vs. zero in the example). Similarly, the CLA sent from LP_2 to LP_1 (CLA_{2-1}) is determined. Furthermore, the condition of the CLA sent from LP_1 to LP_2 breaks if LP_2 sends an external event to LP_1 with time stamp T. In that case, an IA is sent by LP_2 to itself to inform the invalidation of CLA_{1-2} at T. Because T is anyhow after the current time of LP_2, no causality error occurs.

4.2. *Fully-connected topology and line topology*

The approach from the previous subsection can also be applied to the model with more LPs with a *fully-connected topology* (with all LPs directly connected). Figures 2b and 3b show a scenario of a multi-agent system with four LPs and the general model, respectively. When there is direct communication between all LPs and an LA exists between each pair of LPs, each LP can send an identical CLA to all other LPs. This CLA between each LP has the condition that its state will not be modified by any other LP. When the first external event is sent between LPs, an IA with the same timestamp is broadcasted. This ensures that all CLAs from the LP, where the external event is processed, are invalidated. For obvious reasons, the global CLA sent from this LP is shorter compared with any process-to-process CLA from this LP. Thus, irrespective of the destination LP, the timestamp of the external event is larger compared with the global CLA sent from this LP.

The IA is thus subsequent to any other LPs. Thus, there is no causality error caused by the external event and the IA. Reconstruction of the CLAs after the IA is broadcasted is demanding. Thus, the solution with CLA is only practical when the probability of premature CLA invalidation is very small. The mechanism results in an increase in the short LAs, however, possible reduction the long LAs in the system. In this aspect, it leads to a balance of the LAs in the system.

Similarly, this approach is applied to *line topology* models (where all LPs are connected as a chain) as long as the state change of one LP will not cause the immediate state change of a neighboring LP. The notion from the previous section is applicable to each *interface* between a pair of LPs. The calculation of CLA for an interface is performed under the assumption that the state of the LP could be changed any time by a neighboring LP from another interface. Thus, the CLA is limited by the minimal time needed as that state change 'propagated' from one interface to another. For the multi-agent simulation example earlier, it is the minimal time needed for an agent to cross the area simulated by this LP.

Moreover, the approach of CLA is useful for a combination of fully connected topology and chain topology. Here, an interface is shared by a group of LPs organized as a fully connected topology. An example will be illustrated later as a case study. In comparison with the synchronized execution based on conservative LA, more messages will be sent between these LPs to exchange CLAs and IAs. However, no static observation to the whole system is needed, and the LPs will not be blocked due to the observation in general. Our approach is thus convenient especially when the estimated LA is stochastically too conservative. Figure 4 shows a practical implementation of the mechanism with some slight simplification. Here, each LP estimates a global CLA for all neighboring LPs instead of only one CLA per interface. Because it is unknown whether the external event will be processed at the specified LP before sent IAs are processed at some other LPs, serial ID management is necessary for comparison of the number of processed external events and IAs between each pair of LPs.

4.3. *Grouping the logical processes*

The simulation of large models often results in a large number of LPs. For application of the mechanism to these simulations, the LPs should be split into groups based on the transmission delay. This way, the transmission delay between LPs within a group is short and the transmission delay between LPs in different groups is long. For each group, the local synchronized execution is applied and the whole group acts like a single LP. For the synchronization between groups, the solution with CLA is applied, which was discussed in the previous subsections. The cost of such local synchronized execution is much less than that of a global synchronized execution, because the transmission delay within the group is short. However, the local synchronization must be done more frequently for the calculation of CLAs needed for the synchronization between groups. The combination with synchronized execution and CLA is only applicable when the LA is stochastically too conservative, as mentioned earlier.

Alternatively, a synchronization based on conservative LA within each group and synchronization based on CLA between groups can be considered. A scenario with three LPs is shown in Figure 3c. As illustrated in the figure, the three LPs are divided into two groups. LP_1 belongs to the first group, LP_2 and LP_3 belong to the second. Each group is simulated with an individual computer with a sufficient number of processors. Because the transmission delay between LP_1 and LP_2, as well as between LP_1 and LP_3, is very long, we will extend them with CLA. First of all, LP_1 sends CLA to LP_2 and LP_3 (CLA_1) with the condition that its own state will neither be modified by LP_2, nor by LP_3. If LP_2 sends an external event to LP_1 with time stamp T for example, it will send an IA to itself to inform that the CLA_1 is no more valid after T. As discussed earlier, there is no causality error. In addition, it will send the same IA to LP_3. This IA will arrive in time without causality error, if the LA from LP_2 to LP_3 (LA_{2-3}) is shorter than the sum of the LA from LP_2 to LP_1 (LA_{2-1}) and LA from LP_1 to LP_3 (LA_{1-3}). Similarly, it must be guaranteed that the LA from LP_3 to LP_2 (LA_{3-2}) is shorter than the sum of LA from LP_3 to LP_1 (LA_{3-1}) and LA from LP_1 to LP_2 (LA_{1-2}). Otherwise, the mechanism might cause causality errors and cannot be applied. In practice, we only consider LA_{2-1} and LA_{3-1} and ignore the LA_{1-3} and LA_{1-2} because of the extra cost of communication. Most importantly, we limit LA_{2-3} and LA_{3-2} so that it is always

```
1. Initialize
for each neighbor $LP_r$ do
    Let $localID_r = 0$.
    Let $remoteID_r = 0$.
end for

2. Estimate & Send CLA
Let $CLA_{min}$ be the minimal time of state change
propagation from one interface to another.
for each neighbor $LP_r$ do
    Estimate the $CLA_r$ to $LP_r$ with the condition, that
    the local state will not be changed by any other LP
    if $CLA_r < CLA_{min}$ then
        Let $CLA_{min} = CLA_r$.
    end if
end for
for each neighbor $LP_r$ do
    Send $CLA_{min}$ to $LP_r$ with ID $localID_r$.
end for

3. Apply Received CLA
Input: the received CLA $CLA_r$ from $LP_r$ with ID
$id$.
if $id \geq remoteID_r$ then
    Apply $CLA_r$ as the current LA from $LP_r$.
end if

4. Schedule External Event & Broadcast IA
Input: the event $E$ to $LP_r$ with time stamp $T$.
Send $E$ to $LP_r$.
for each $LP_t$ in any fully-connected topology with
the local LP and $LP_r$ do
    Send IA to $LP_t$ with time stamp $T$ and parameter
    $LP_r$.
end for

5. Process Received External Event
Input: the event $E$ from $LP_r$.
Process $E$.
for each $LP_t$ in any fully-connected topology with
the local LP and $LP_r$ do
    $localID_t$++.
end for
Do 2. to estimate and send the new CLA

6. Process Received IA
Input: the IA from $LP_r$ with parameter $LP_t$.
Set the current LA from $LP_r$ to 0.
$remoteID_r$++.
```

Figure 4. Pseudocode of the algorithm.

shorter than LA_{2-1} and LA_{3-1}. Furthermore, LP_2 sends CLA to LP_1 with the condition that LP_1 will not modify the state of LP_2 and LP_3 with external events. If LP_1 sends an external event to LP_2, an IA will be sent to itself as discussed previously.

This alternative solution can be seen as a trade-off between the external LAs and internal LAs. On one side, any LA sent from LP_i to any LP in the same group is limited by

$$\min\{LA_{i-k} | LP_k \text{ belongs to the other group}\}.$$

Conditional look-ahead (CLA) can be applied to extend these LAs anyway, but the limit must be kept. On the other side, the LA / CLA sent from LP_i to any LP in the other group could be extended by a condition. Whether this condition really helps to extend the LA depends strongly on the model and the partitioning. This is generally unpredictable due to the unknown state of LPs in the same group. But in some special situations, the interruption of some immediate communication loops is useful.

5. EVALUATION

We will evaluate our solution based on the exchange of conditional look-ahead here. We consider a set of multi-agent systems, because these scenarios show the characteristics suitable for our approach.

5.1. Environment and estimation algorithms

All models for the evaluation are implemented with our simulator named *Universal Simulation Engine (USE)* [21] [‡]. We shall consider the 'simulation of simulation' for study. In other words, we simulate a distributed simulator of the given model. Other than the model itself, we also simulate core behaviors of the distributed simulator. These include the transmission delay, cost to lock and unlock an LP, the execution power of each LP, wall-clock time of the virtual simulator, and so on. The result represents the simulation result with nine individual computers.

Universal Simulation Engine (USE) applies a framework of conservative synchronization to adhere to causality by using the *process-to-process barriers (P2P barriers)*. A barrier represents a specific time bound that the process should never reach. During the simulation, each process should set a barrier to every process, including itself based on the predictable LAs. These barriers will be altered (moved back) by a specified algorithm. The framework does not include the algorithm for calculation and alteration of barriers. However, many ideas exist, which can be slightly modified and applied to this framework.

The most popular algorithms for synchronized execution, known as the barrier synchronization algorithms, are summarized in [1]. The Butterfly Barrier algorithm [20] is implemented as follows for the evaluation:

(1) First, each process sets proper barriers to all other processes. After process P_i has received these barriers, it calculates which events are safe to process. In addition, it can calculate the virtual time t_i for the next synchronization, as the minimum of all received barriers.
(2) Each process P_i sets the barriers that are received from the processes, with which it should synchronize later to t_i. For example, if there are eight processes, the process P_0 will synchronize with process P_1, P_2, and P_4. Furthermore, it sets its own barrier to t_i.
(3) When a process needs to convey a signal to another process for synchronization, it alters the barrier to that process to infinity. When a process P_i is blocked, it checks the barriers received from other processes within the call-back function. If the relevant barrier is larger than t_i, then the signal has been received by a process. Then the process P_i sends signals back to these processes with respect to the synchronization algorithm. If all signals cannot be sent at that point in time, the process goes to wait.
(4) If a signal is received during the wait period, the process will be activated and will go to wait again, because one of the barriers at the present time point has been altered back. The call-back function will be triggered for another time to re-check the situation and to send more signals.
(5) After all signals have been received by P_i, the process executes the barrier primitive and sets new barriers to the other processes. The new barrier must be however larger than t_i. Then, another period will be started.

5.2. The models

As a case study, we consider the simulation of a multi-agent system build on *Universal Multi-Agent System (UMAS)*[§], a framework of multi-agent models based on our simulator USE. An agent of UMAS framework is always under a *movement* with a given speed and a duration. A waiting period will be considered as movement with zero speed. After the end of a movement, the agent arrives at its *destination* and another movement will be started immediately. Movement in UMAS cannot be canceled.

[‡]The simulator is available at http://use.useproject.com/.
[§]The framework is available at http://umas.useproject.com/.

To calculate the CLA, the current movement of each agent is analyzed. If the destination (location after the end of current movement) is located in the range of another LP, the exact time that the agent crosses the border line is calculated as the *hand-over time* T_h. Otherwise, we calculate

$$T'_h = T_e + D_{\min}/S_{\max}$$

as the *potential hand-over time*, where T_e is the end time of current movement, $D_{\min}$ is the minimal distance from the destination to border line, and $S_{\max}$ is the maximal speed of the agent. Then, we calculate

$$T_{h,\min} = \min\{\min\{T_h\}, \min\{T'_h\}\}$$

as the minimal hand-over time (the bounding time stamp). The difference between the minimal hand-over time and current time represents the global CLA, which will be sent to neighboring LPs after the calculation.

As mentioned in our algorithm (Figure 4), the CLA will be broadcasted after an external event is received (an agent is handed in). If there is no external event, the CLA will be broadcasted periodically. To calculate the time of next broadcasting, we consider

$$T_{e,\min} = \min\{\min\{T_e\}, T_{h,\min}\}$$

as the *end time of the next movement* in the system. Obviously, $T_{h,\min}$ will not advance before the time defined by $T_{e,\min}$. In other words, no new minimum of hand-over time can be calculated before the end of the next movement. Thus, a new calculation of $T_{h,\min}$ before $T_{e,\min}$ is pointless. However, a new calculation of $T_{h,\min}$ must be done within the time-frame defined by current $T_{h,\min}$, because the neighboring LPs will be blocked after that time (expiration of CLA). In UMAS, a new $T_{h,\min}$ will be calculated at the time

$$T_c = (T_{h,\min} + T_{e,\min})/2$$

if no external event is received.

An example is shown in Figure 5 where three agents are located in the range of the LP, and we consider that no agent will be handed in the short future. At the current time, the CLA is being calculated, and we analyze the current movement of the three agents. The destination of agent 1 and agent 3 are still located in the range of the LP. We calculate the end time of the current movement (time C), plus the minimal time from the destination to border line (time A). The destination of agent 2 is out of range of this LP; a hand-over is scheduled during the movement (time B). The minimal hand-over time is calculated as the minimum of time A for agent 1, time B for agent 2, and time A for agent 3. Furthermore, we calculate the minimum of time C for agent 1, 2, and 3 as the end time of the next movement. Subsequently, T_c is calculated. The CLA will be re-calculated and broadcasted at that time.

As an example, we consider the simulation of a multi-agent system. The whole area is divided into nine sub-areas (3 × 3) as shown in Figure 6a, and each sub-area will be simulated with one LP.

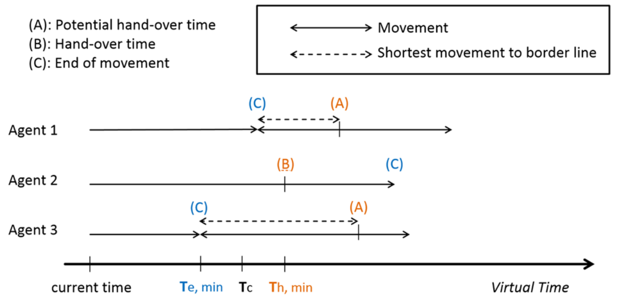
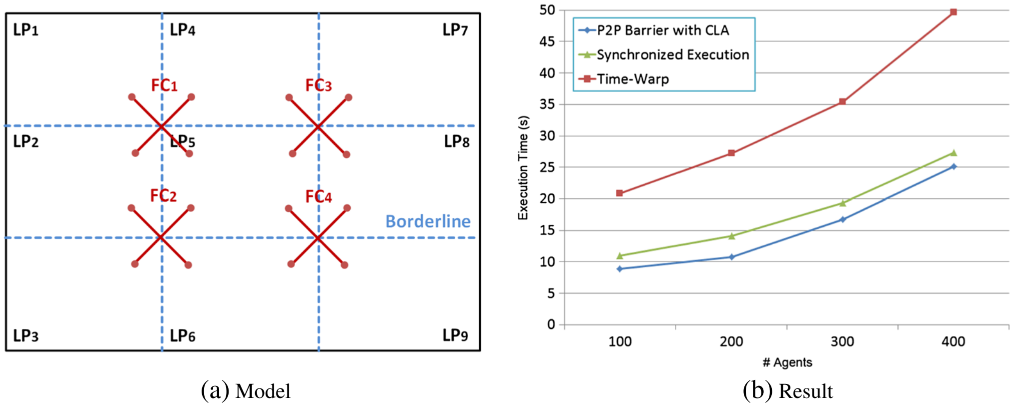


Figure 5. Example: periodical broadcasting of conditional look-ahead.



(a) Model (b) Result

Figure 6. Case study.

There exist n agents in the whole area, which are positioned randomly at the start with a random direction. They move straight and forward with a constant speed. When an agent reaches a border of the whole area, it will be rebounded. However, a random duration is chosen for each movement. The simulator itself holds no extra knowledge about the intelligence in the agent and the future movement is unpredictable. Moreover, each agent saves information about the agent's 'meetings' (distance is short enough) as well the timestamp of these meetings.

5.3. *Results*

The scenario to be simulated is considered as a combination of four fully connected topologies and several line topologies between the stars. For example, the fully connected topology FC_1 contains LP_1, LP_2, LP_4, and LP_5. If LP_1 sends an external event to LP_4, it will broadcast an IA to LP_2, LP_4 ('included' in the external event), and LP_5. Furthermore, LP_4 belongs to two different fully-connected topologies (FC_1 and FC_3). In other words, it has two interfaces. The state changes of LP_4 due to the LP_1 will not cause immediate state change of LP_7, for example, because of the minimal time needed for an agent to cross the sub-area simulated by LP_4. In many models of multi-agent system including this one, the minimal time needed that the state change propagated from one interface to another interface of a LP is longer than any CLA provided by the LP, and it will be thus ignored.

The LPs are synchronized with the algorithm based on CLA and IA. In comparison, we also consider the standard time-warp algorithm as well as the globally synchronized execution based on Butterfly Barrier synchronization [20]. The result of the simulation is shown in Figure 6b. The simulation is more time consuming when there are more agents to be simulated. The execution time for time warp is very great due to the extra cost involved with saving the state of each agent and its AI. Obviously, this time-warp algorithm can be further optimized. But this is out of the scope of our discussion. The results of global synchronized simulation and the results based on CLA are comparable. Our approach takes about 10–24% less time compared with the approach using globally synchronized execution, depending on the number of agents in the system.

6. CONCLUSION

We presented a new mechanism to accelerate DDES using the exchange of the CLA and broadcast of IA. Our approach is able to increase the LA between LPs, especially when the estimated LA is stochastically too conservative and the probability that an external event really occurs right after the expiration of the LA, is very small. The basic idea of our approach is to exchange information between neighboring LPs for the dynamic conservative estimation of the LA. This approach is a trade-off between an estimation based on the global state and an estimation based on the local state.

This way, a better estimation of the LA is achieved at lower cost. The approach is applicable to various existing models without significant modification, because the only model-dependent parameters concern the exchange of CLA and IA. However, the algorithm to estimate the LA has to be modified to calculate the CLA instead of the LA.

We also presented two typical topologies where CLA is applied to accelerate the simulation. Firstly, if all LPs are allowed to communicate with each other directly (fully-connected topology), then each LP sends an identical CLA to all other LPs. This CLA is with the condition, that its local state will not be modified by any other LP. The broadcast of IA within the whole system occurs after any external event is sent. Additionally, the approach can be applied to a chain of LPs (line topology), given that the state change of one LP will not cause the immediate state change of a neighboring LP. Most importantly, the approach also works for a combination of both topologies. We also showed the outline of applying the solution with CLA to more complex models by grouping of LPs. Our case studies of multi-agent systems based on UMAS and USE showed that the execution time of the simulation is reduced by up to 24% compared with the solution based on synchronized execution.

REFERENCES

1. Fujimoto RM. *Parallel and Distribution Simulation Systems* (1st ed.) John Wiley & Sons Inc.: New York, NY, USA, 1999.
2. Niewiadomska-Szynkiewicz E, Sikora A. Algorithms for distributed simulation - comparative study. In *Parallel Computing in Electrical Engineering, 2002. PARELEC '02. Proceedings. International Conference on*, 2002; 261 – 266.
3. Fu D, Becker M, Szczerbicka H. On the potential of semi-conservative look-ahead estimation in approximative distributed discrete event simulation. In *Proceedings of the 2013 Summer Computer Simulation Conference, Ser. SCSC '13. Vista, CA: Society for Modeling & Simulation International*, 2013; 28:1–28:8.
4. Leye S, Uhrmacher A, Priami C. A bounded-optimistic, parallel beta-binders simulator. In *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*, Vancouver, BC, Oct 2008; 139–148.
5. Bryant RE. Simulation of packet communication architecture computer systems. *Technical Report,* Massachusetts Institute of Technology, Cambridge, MA, USA, 1977.
6. Chandy K, Misra J. Distributed simulation: a case study in design and verification of distributed programs. *Software Engineering, IEEE Transactions on* 1979; **SE-5**(5):440–452.
7. Nicol DM. Parallel discrete-event simulation of fcfs stochastic queueing networks. *SIGPLAN Not.* 1988; **23**(9): 124–137.
8. Lin Y, Lazowska E. Exploiting lookahead in parallel simulation. *IEEE Transactions on Parallel and Distributed Systems* 1990; **1**(4):457–469.
9. Meyer RA, Bagrodia RL. Improving lookahead in parallel wireless network simulation. In *Proceedings of the 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Ser. MASCOTS '98*, IEEE Computer Society, Washington, DC, USA, 1998; 262–267.
10. Liu J, Nicol DM. Lookahead revisited in wireless network simulations. In *Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation, Ser. PADS '02*, IEEE Computer Society, Washington, DC, USA, 2002; 79–88.
11. Jefferson DR. Virtual time. *ACM Transactions on Programming Languages and Systems* 1985; **7**(3):404–425.
12. Martini P, Rümekasten M, Tölle J. Tolerant synchronization for distributed simulations of interconnected computer networks. *SIGSIM Simulation Digest* 1997; **27**(1):138–141.
13. Ferscha A, Chiola G. Self-adaptive logical processes: the probabilistic distributed simulation protocol. In *Simulation Symposium, 1994., 27th Annual*, La Jolla, CA, 1994; 78–88.
14. Ferscha A. Probabilistic adaptive direct optimism control in time warp. In *Parallel and Distributed Simulation, 1995. (PADS'95), Proceedings., Ninth Workshop on (Cat. No.95TB8096)*, Lake Placid, NY, June 1995; 120–129.
15. Kunz G, Stoffers M, Gross J, Wehrle K. Know thy simulation model: analyzing event interactions for probabilistic synchronization in parallel simulations. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques, Ser. SIMUTOOLS '12*, ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012; 119–128.
16. Park H, Fishwick P. A fast hybrid time-synchronous/event approach to parallel discrete event simulation of queuing networks. In *Simulation Conference, 2008. WSC 2008. Winter*, Austin, TX, December 2008; 795–803.
17. Fu D, Becker M, Szczerbicka H. Accelerating distributed discrete event simulation through exchange of conditional look-ahead. In *Distributed Simulation and Real Time Applications (DS-RT), 2014 IEEE/ACM 18th International Symposium on*, IEEE, Toulouse, 2014; 183–189.
18. Jafer S, Wainer G. Global lookahead management (GLM) protocol for conservative devs simulation. In *Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on*, Fairfax, VA, October 2010; 141–148.

19. Chetlur M, Wilsey P. Causality and proactive cancellation. In *Distributed Simulation and Real-Time Applications, 2006. DS-RT'06. Tenth, IEEE International Symposium on*, IEEE, Terremolinos, 2006; 193–200.
20. Nicol DM. Noncommittal barrier synchronization. *Parallel Computing* 1995; **21**(4):529–549.
21. Fu D, Becker M, Szczerbicka H. Universal simulation engine (use) - a model-independent library for discrete event simulation. *The Spring Simulation Multi-Conference, 2015. SpringSim'15. Annual Simulation Symposium, 2015 ANSS'15*, San Diego, CA, USA, 2015; 146–154.