*Proceedings of the 2008 Winter Simulation Conference*
*S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.*

# IMPROVING PERFORMANCE BY REPLICATING SIMULATIONS WITH ALTERNATIVE SYNCHRONIZATION APPROACHES

Zengxiang Li
Wentong Cai
Stephen John Turner
Ke Pan

School of Computing Engineering
Nanyang Technological University
Singapore 639798

## ABSTRACT

Parallel and distributed simulation facilitates the construction of a simulation application (i.e., federation in HLA terminology) with a number of simulation components (federates). Recently, an approach based on active replication technique has been proposed to improve the performance of simulations by exploring software diversity. To guarantee the correctness of the approach, all replicas of the same federate are required to be Piece-Wise-Deterministic (PWD). However, the PWD restriction will not be satisfied if the replicas achieve software diversity by employing different kinds of synchronization approaches. A replication structure is proposed in this paper, which can transparently handle the federate replicas that use either conservative or optimistic synchronization approach. The execution speed of the federation is increased by always choosing the fastest replica to represent the federate in the federation. Besides presenting the implementation details, we also report the experimental results to demonstrate the performance improvement of the replication structure.

## 1 INTRODUCTION

Parallel and distributed simulation facilitates the construction of a simulation application (federation) with a group of simulation components (federates), which can be developed independently and executed on either a computation environment with parallel processors or geographically distributed computing resources. In many problem domains, simulation applications are time consuming and computation intensive, e.g., traffic simulation, supply chain simulation and simulation of computer networks. Hence, speedup of simulation execution is important. Recently, the active replication technique has inspired a novel direction to enhance the performance of simulation. Generally, more resources are needed in an active replication scheme, since each federate has multiple replicas. However, due to the rapid development of Grid technology (Foster et al. 2001) during the last decade, the resources can be obtained more and more easily.

A typical active replication approach has been proposed in (Quaglia 2006) to increase the execution speed of a federation by employing software diversity in the replication of federates. Software diversity can be achieved by employing different data structures, libraries and algorithms in the development of multiple programs that are functionally the same. In (Quaglia 2006), multiple replicas are developed for the same federate using different state saving mechanisms. An Active Replication Management Layer (ARML) is developed to support the concurrent execution of these replicas. ARML is a kind of message proxy layer. For a message sent by a federate, ARML only sends other federates in the federation the first instance of the message generated by the replicas of the federate; for a message destined for a federate, ARML distributes it to all replicas of the federate in the correct order and buffer the message if necessary. The performance of a federation is improved, because the simulation time of a federate is always advanced by the fastest running replica. However, ARML requires Piece-Wise-Deterministic (PWD) is satisfied by all replicas of the same federate to guarantee the correctness of the simulation execution. That is, all replicas must execute events in the same order and send out the same messages following the same sequence; and messages are also delivered to all replicas of the federate following the same sequence.

The other well studied direction to improve the performance of parallel and distributed simulation is to reduce the synchronization overhead. Two kinds of synchronization approaches are commonly used: conservative (e.g., CMB protocol (Bryant 1977, Chandy and Misra 1979)) and optimistic (e.g., Time Warp protocol (Jefferson 1985)) approaches. Conservative approaches preserve the causality constraint by ensuring that each federate processes events in strictly non-decreasing order. In contrast, optimistic approaches allow free processing of events and thus potentially have better exploitation of parallelism among federates. However, some additional mechanisms are needed to recover a federate from a causality error.

A great amount of research has been done on evaluating the performance of conservative approaches and optimistic approaches (Niewiadomska-Szynkiewicz and Sikora 2002). The efficiency of a conservative approach is critically reliant on lookahead extraction, which is simulation model dependant. An optimistic approach may outperform a conservative one if causality errors seldom occur. Therefore, different simulation applications may need to use different synchronization approaches to achieve higher efficiency. Even for the same simulation application, different inputs and parameters may lead to different suitable synchronization approaches. It may also happen that some federates in the federation are executed efficiently using conservative approach, while others are executed efficiently using optimistic approach. Moreover, the suitability of a synchronization approach for the same federate may change dynamically. A number of protocols have been proposed to reduce the synchronization overhead via taking advantages of both conservative and optimistic approaches. These will be further discussed in section 2.

Inspired by the concept of active replication and employing the diversity of synchronization approaches, we propose to enhance the performance of a federation by concurrently running multiple replicas with alternative synchronization approaches for the same federate. These replicas are likely to exhibit different runtime performances, as different synchronization approaches may perform diversely for the same federate. The performance is enhanced by always choosing the fastest replica to represent the behavior of the federate in the federation. As PWD cannot be guaranteed when replicas use different synchronization approaches, the ARML proposed in (Quaglia 2006) cannot be used. In this paper, a replication structure is introduced, which can transparently handle the federate replicas that employ either conservative or optimistic approach.

Some background knowledge that is necessary for the understanding of our replication structure is introduced in section 3. Section 4 presents the implementation details of our method to replicate federates with alternative synchronization approaches. Experimental results are reported and discussed in section 5. Finally, section 6 concludes the paper and outlines the future work.

## 2 RELATED WORK

As mentioned above, a conservative approach cannot outperform an optimistic approach in every situation, and vice versa. Since the beginning of 1990s, a number of protocols have been proposed to obtain higher efficiency by taking advantages of both conservative and optimistic approaches. These protocols can be classified into two groups as follows.

*Find the compromise between conservative and optimistic approaches:* conservative and optimistic approaches go towards two extremes: conservative approaches impose strict constraint on simulation components in simulation time advancement and event processing; whereas optimistic approaches allow free processing of events with the price of error recovery. To reach the compromise, one can either introduce optimism in a conservative approach (e.g., the optimistic-conservative synchronization proposed by Xu and McGinnis (2006)), or introduce conservatism in an optimistic approach (e.g., Moving Time Window protocol (Sokol et al. 1988) and Adaptive Time Warp protocol (Ball and Hoyt 1990)).

*Mix and switch between different synchronization approaches:* the Local Time Warp method proposed by (Rajaei et al. 1993) enables mixing of synchronization approaches hierarchically in a simulation. At the lower level, an optimistic approach is used to exploit parallelism, while a conservative approach is adopted to control cascaded rollbacks at the higher level. The parallel simulation protocol proposed by Jha and Bagrodia (1994) not only allows different simulation components to apply different synchronization approaches, but also supports the simulation components to switch among different approaches dynamically.

Different from the above existing approaches, our proposal allows federate replicas to employ alternative synchronization approaches. The replication structure can transparently support the concurrent execution of federate replicas and automatically choose the fastest replica to represent the federate. In other words, alternative synchronization approaches are enabled simultaneously for a federate, and the federate can switch among these approaches automatically, depending on their runtime performance.

## 3 HLA AND SOHR ARCHITECTURE

The replication structure is developed for High Level Architecture (HLA) based simulations and implemented in a totally transparent manner by exploiting the decoupled federate architecture in SOHR (Pan et al. 2007), a service-oriented HLA RTI.

### 3.1 Time Management in HLA

The HLA was developed initially by the DMSO (Defense Modeling and Simulation Office ), and was adopted as IEEE 1516 standard (IEEE 2000) in September 2000. HLA provides a general architecture for developing large-scale distributed simulation (federation) and supports the reusability and interoperability of simulation components (federates). The Runtime Infrastructure (RTI) communication middleware is an implementation of the HLA interface specification. Federates in the same federation can only communicate with each other through the underlying RTI.

A group of time management (TM) services are defined in HLA to control the time advancement of each joined federate along the HLA time axis (IEEE 2000). There are two types of messages in a federation: *Time-Stamp Order* (TSO) and *Receive Order* (RO) messages. Federates can be divided into four categories based on their *time regulating and time constrained* status. The federates discussed in this paper are both time regulating and constrained, among which only TSO messages are exchanged.

The *Next Messages Request* (NMR) service with parameter *t* is intended to be used for conservative synchronization of event driven federates, where *t* is typically the time stamp (TS) of the next local event. A *Time Advance Grant* (TAG) shall complete this request. The federate is granted to *t* and no TSO message is delivered, if *t* is smaller than the minimum TS of the TSO messages that will ever be received by the federate. Otherwise, the federate is granted to this minimum TS, and the TSO messages with TS equals to the minimum TS are eligible to be delivered to the federate.

Optimistic synchronization is supported by *Flush Queue Requests* (FQR) service. It forces the RTI to deliver all TSO messages. An FQR can always be granted without waiting for other federates to advance. The granted time is the upper bound for fossil collection, which can be smaller than the current simulation clock of the federate. When causality error happens, an optimistic federate can use the *Retract* service to retract a perviously sent message. If the message has already been delivered to a federate, the RTI will invoke the *Request Retraction* callback to inform the federate to remove the message. This may cause the federate to rollback.

As for message exchange among federates in the federation, *Send Interaction* (SI) service is used to send an interaction to remote federates, while *Update Attribute Values* (UAV) service is used to supply object instance attribute values to other interested federates. The messages are received by the federate invoking the *Receive Interaction* (RI) or the *Reflect Attribute Values* (RAV) callbacks.

### 3.2 SOHR Architecture

SOHR is a service-oriented architecture which provides the functionalities of the HLA RTI via the cooperation of a group of predefined Grid services (Foster et al. 2001). The overview of the architecture is shown in Figure 1. HLA RTI Management Services implement the six service groups defined by the HLA interface specification. The RTI Index Service provides a system-level registry of all other Grid services.

In SOHR, a federate is decoupled into two components: federate simulation model and Local Service (LS). The design of LS follows the WS-Resource factory design pattern. It consists of factory service LFS, instance service

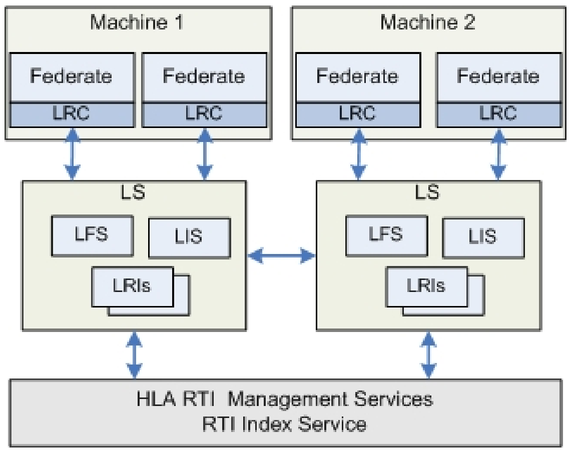


Figure 1: SOHR architecture.

LIS and multiple resource instances LRIs. LFS creats a LRI when a federate joins a federation and destroys the LRI when the federate resigns from the federation. The LRI is a stateful class that is used to keep the RTI state concerned with the federate, such as TM information and TSO/RO callbacks. The LIS provides service to the federates by accessing the LRI state and communicating with the HLA RTI Management Services and LISes of other federates. The RTI services initiated by the federate are translated into Grid service invocations of the LIS by the LRC (Local RTI Component). When the LIS receives a message from other LISes, a corresponding callback will be generated and saved in a message queue depending on the message type. Finally, federate requests for callbacks by calling Evoke Multiple Callbacks (IEEE 2000), and LRC obtains callbacks from the LRI by invoking the *getcallbacks()* service of the LIS and delivers the callbacks to the federate. In summary, the federate only deals with the local computation; whereas the LIS/LRI handles the distributed part, playing the role of the LRC in the traditional HLA RTI implementations (e.g., RTI NG (Defense Modeling and Simulation Office )).

## 4 REPLICATING FEDERATES WITH ALTERNATIVE SYNCHRONIZATION APPROACHES

### 4.1 Replication Structure

The replication structure in SOHR is shown in Figure 2. Multiple replicas, which are developed for the same federate, can be executed concurrently. These replicas can employ either a conservative approach or an optimistic approach, which are denoted as CON and OPT respectively. Due to the decoupled federate architecture in SOHR, it is the LIS/LRI instead of the federate replicas that connects to the federation directly. Therefore, multiple replicas of the same federate can connect to a single LIS/LRI, while the LIS/LRI takes care of time advancement requirement and message exchanges with other federates. The LIS/LRI can transparently support the concurrent execution of federate replicas and improve the performance by always choosing the fastest replica to connect to the federation, i.e., communicating with the LIS/LRIs of other federates and the HLA RTI management services.

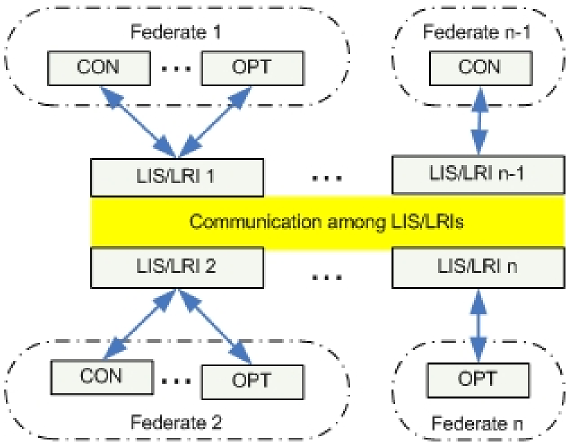


Figure 2: Replication structure.

The replication structure is designed in a totally transparent manner. Firstly, the replication of a federate is transparent to the federate itself. The federate replica does not know the existence of other replicas of the same federate. Secondly, the replication of a federate is transparent to remote federates. They have no idea about which federate replica they are actually communicating with.

The LIS/LRI plays a similar role as ARML, and thus there is no need to insert an additional layer between federate and RTI. Unlike ARML the PWD restriction is no longer required in our replication structure. CON and OPT behave differently due to the characteristics of the employed synchronization approaches. They use different RTI services to request time advancement. Moreover, OPT may process events in wrong order and incorrect messages might be generated as a result. To maintain its consistency with CON, OPT needs to rollback to a correct sate and retract the incorrect messages which were generated previously. Therefore, to correctly deal with these different behaviors between CON and OPT, the LIS/LRI in SOHR should be modified to:

1. Filter the messages sent by both CON and OPT and deliver only the first instance of the message to the subscribers;
2. Handle the additional messages sent by OPT, including the incorrect messages and the corresponding *Retract* service;

Table 1: Additional variables and flags

| Name | Description |
|---|---|
| ConLGT | Last Granted Time of CON |
| OptLGT | Last Granted Time of OPT |
| LGT | Max(ConLGT, OptLGT) |
| ConReqTime | Requested time from CON |
| OptReqTime | Requested time from OPT |
| ConInTAS | CON is in time advancing state |
| OptInTAS | OPT is in time advancing state |

3. Deliver eligible TSO callbacks to CON and OPT, according to the HLA interface specification;
4. Prevent any incorrect message and *Retract* service to be delivered to the CON replica of the subscribers; and
5. Handle time advancement request from CON and OPT and grant time request to them individually.

To perform the above tasks, the LIS/LRI should have two major functions: Firstly, it must guarantee the correctness and efficiency of message exchanges among federates. Secondly, it must coordinate correctly the execution of conservative and optimistic replicas of the same federate. Currently, the replication structure is implemented to support two replicas of the same federate: one is CON while the the other is OPT. They are assumed to use NMR and FQR respectively to request time advancement. It is also assumed that the two replicas have the same value of lookahead (denoted as LA).

Some additional variables and flags are defined in the LIS/LRI, which are shown in Table 1. ConLGT represents the last granted time of CON, which defines the upper bound of the logical time to which CON can advance. OptLGT represents the last granted time of OPT, which defines the lower bound of logical time of any future rollbacks. ConLGT and OptLGT may have different values, as their calculation depends on the relative performance of CON and OPT (as to be discussed in section 4.3). LGT represents the logical time of the federate, which is maintained by the LIS/LRI. Since the faster federate replica is chosen to represent the federate, LGT is set to the bigger value between ConLGT and OptLGT.

## 4.2 Message Exchange

The mechanism of message exchange among replicated federates is shown in Figure 3. The solid lines represent the trace of outgoing messages from the federate, while the dashed lines show the flow of incoming messages to the federate.

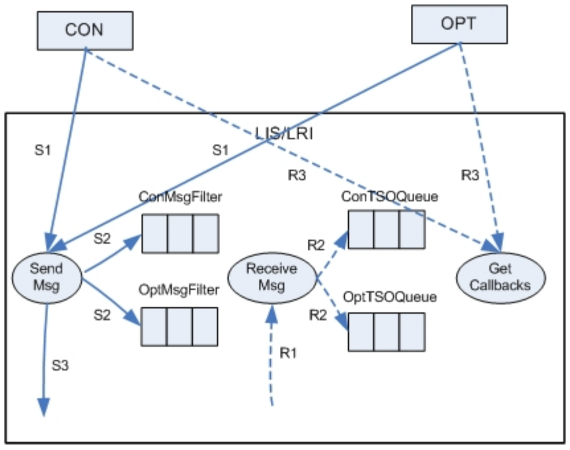


Figure 3: Mechanism of message exchange.

In the replication structure, outgoing messages can be generated by both CON and OPT (step S1). Different messages might be generated by CON and OPT in different order. When a message is generated by one replica, the LIS/LRI needs to determinate whether the message has already been generated by the other replica. Two message filters are deployed for CON (ConMsgFilter) and OPT (OptMsgFilter) to buffer the messages that have been sent by the LIS/LRI to other federates respectively. After a message is generated by a federate replica, the LIS/LRI will search the same instance of the message in the message filter of the other replica (step S2). If the same instance is found, which indicates the message has been sent, the message is ignored and the same instance of the message is removed from the message filter of the other replica (step S3). Otherwise, the message is sent out by the LIS/LRI and buffered in the message filer of the replica (step 3).

As LGT is the logical time of the federate, the LIS/LRI will never send out any messages with TS less than LGT + LA. Thus, the generated message can be ignored by the LIS/LRI, if its TS is smaller than LGT+LA. In this way, many comparisons are avoided and the message filters can be reclaimed when LGT is updated. The comparison overhead is also reduced by sorting the messages in the message filters by TS.

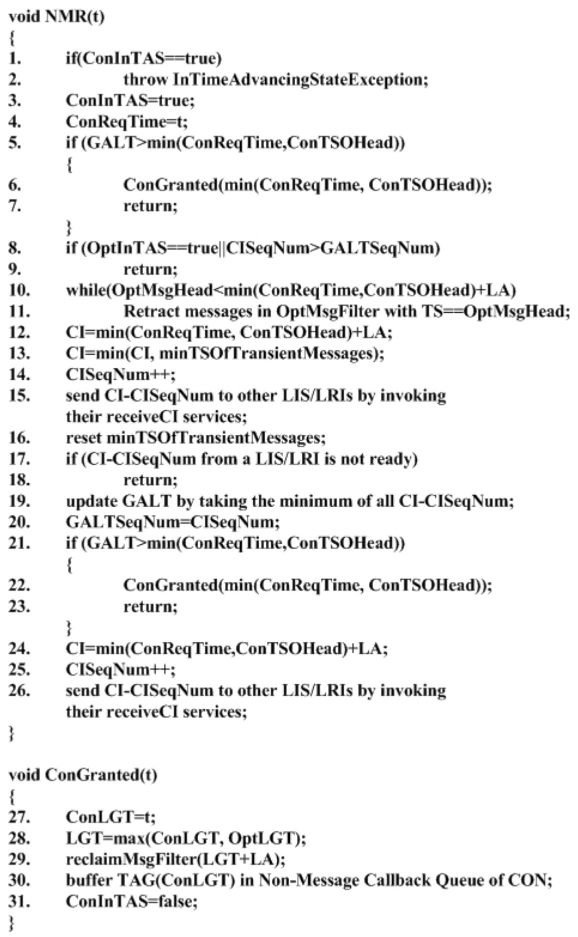
As described above, for a message sent by both CON and OPT, the first instance of the message is sent out to the subscribers, while the second one is simply ignored by the LIS/LRI. The message generated only by OPT must be an incorrect message and should be retracted latter by OPT using the *Retract* service. Therefore, an incorrect message can be identified, if the LIS/LRI can ensure that the CON replica never ever sends the same instance of the message. The incorrect message, if identified, is simply ignored by the LIS/LRI. Otherwise the message generated by OPT is sent out to the subscribers and buffered in the OptMsgFilter. When handling the *Retract* service invoked by OPT, the LIS/LRI checks whether the corresponding message is kept in the OptMsgFilter. If not, the *Retract* service is ignored simply. Otherwise, the LIS/LRI sends retraction of the message to the subscribers and removes the message from the OptMsgFilter. In this way, the incorrect messages sent by OPT are handled properly. They and their corresponding retraction requests are either ignored or sent to the subscribers.

To deliver eligible TSO callbacks to CON and OPT, two individual TSO queues are deployed in the LIS/LRI: one for CON (ConTSOQueue), and the other for OPT (OptTSOQueue). As shown in Figure 3, after a TSO message is received (step R1), a corresponding TSO callback is generated and buffered in both TSO queues (step R2). Federate replicas get TSO callbacks from their own TSO queue according to the HLA interface specification (step R3). All TSO callbacks in OptTSOQueue can be delivered to OPT; whereas only the TSO callbacks with TS equal to ConLGT can be delivered to CON. Similarly, two individual Non-message Callback Queues (not shown in the figure) are deployed in the LIS/LRI: one for CON to buffer the *TAG* callbacks, and the other for OPT to buffer the *TAG* and *Request Retraction* callbacks. Hence, the LIS/LRI can grant the time request from CON and OPT individually and avoid delivering *Request Retraction* callbacks to CON.

## 4.3 TM in Replication Structure

A synchronous TM algorithm has been proposed in (Pan et al. 2008) to control the time advancement of the federates along the federation time axis. Federates discussed in that algorithm have only a single instance. Each federate propagates Conditional Information Report (CI) (Fujimoto 2000) with a sequence number (denoted as CI-CISeqNum) to all other federates in the federation. The Greatest Available Logical Time (GALT) is calculated as the minimum value of all CIs with the same sequence number from all federates in the federation. The synchronous TM algorithm proposed in (Pan et al. 2008) is modified to handle the NMR and FQR requests from the replicas. the CISeqNum and GALT are managed by the LIS/LRI. The CI is sent and GALT is calculated by the LIS/LRI when it handles NMR or FQR service from replicas or receives CI from other LIS/LRIs.

The LIS/LRI will send out a CI-CISeqNum when the request of time advancement from either CON or OPT cannot be granted by the current value of GALT. Meanwhile the sequence number of CI (CISeqNum) will be increased by one. This conditional information guarantees that the LIS/LRI will not send out any additional TSO messages or retraction of message with TS less than CI, if it does not receive any new TSO message or retraction of message with TS less than CI − LA. After the CIs with the same sequence number are received from all LIS/LRIs, the GALT can be calculated as the minimum value of these CIs. The GALT defines the lower bound of TS of messages or retraction of messages that the LIS/LRI may receive in the future. By using this value, the LIS/LRI can handle the NMR and FQR services according to the HLA interface specification and grant the time advancement request to CON and OPT accordingly. In general, the LIS/LRI is likely to send out CI and increase the value of GALT while handling the time advancement request from the faster federate replica. With the increased GALT, the time advancement request from the slower replica will be granted directly.



```
void NMR(t)
{
1.      if(ConInTAS==true)
2.              throw InTimeAdvancingStateException;
3.      ConInTAS=true;
4.      ConReqTime=t;
5.      if (GALT>min(ConReqTime,ConTSOHead))
        {
6.              ConGranted(min(ConReqTime, ConTSOHead));
7.              return;
        }
8.      if (OptInTAS==true||CISeqNum>GALTSeqNum)
9.              return;
10.     while(OptMsgHead<min(ConReqTime,ConTSOHead)+LA)
11.             Retract messages in OptMsgFilter with TS==OptMsgHead;
12.     CI=min(ConReqTime, ConTSOHead)+LA;
13.     CI=min(CI, minTSOfTransientMessages);
14.     CISeqNum++;
15.     send CI-CISeqNum to other LIS/LRIs by invoking
        their receiveCI services;
16.     reset minTSOfTransientMessages;
17.     if (CI-CISeqNum from a LIS/LRI is not ready)
18.             return;
19.     update GALT by taking the minimum of all CI-CISeqNum;
20.     GALTSeqNum=CISeqNum;
21.     if (GALT>min(ConReqTime,ConTSOHead))
        {
22.             ConGranted(min(ConReqTime, ConTSOHead));
23.             return;
        }
24.     CI=min(ConReqTime,ConTSOHead)+LA;
25.     CISeqNum++;
26.     send CI-CISeqNum to other LIS/LRIs by invoking
        their receiveCI services;
}

void ConGranted(t)
{
27.     ConLGT=t;
28.     LGT=max(ConLGT, OptLGT);
29.     reclaimMsgFilter(LGT+LA);
30.     buffer TAG(ConLGT) in Non-Message Callback Queue of CON;
31.     ConInTAS=false;
}
```

Figure 4: Implementation details of NMR.

The pseudo code of NMR is shown in Figure 4. ConTSOHead refers to the minimum TS of messages in the ConTSOQueue, and OptMsgHead denotes the minimal TS of the messages in the OptMsgFilter. When the NMR is invoked by CON, GALT is compared with ConReqTime and ConTSOHead to check whether a TAG can be delivered to CON directly (Lines 5-7). If not, the LIS/LRI will send a CI with CISeqNum to LIS/LRIs of other federates (Lines 12-15). In the case where the LIS/LRI has received CI-CISeqNum from LIS/LRIs of all other federates, the value of GALT can be calculated by taking the minimum value of all CI-CISeqNum, including the one sent by the LIS/LRI itself (Lines 17-20). After the GALT has been updated, the LIS/LRI will check again whether a TAG can be delivered to CON (Lines 21-23). If not, CON remains in time advancing state and the LIS/LRI has to send out another CI and carry out a new round of GALT calculation (Lines 24-26). When a TAG is delivered to CON (Lines 27-31), ConLGT and LGT must be updated. Meanwhile, both ConMsgFilter and OptMsgFilter are reclaimed by removing the buffered messages with TS smaller than LGT+LA.

Before sending out a CI, the LIS/LRI has to guarantee that the other replica is not in time advancing state (Line 8). In the case where the other replica is in time advancing state, which implies GALT calculation is in progress, it is unnecessary for the replica to send out another CI and calculate the value of GALT. In the synchronous TM algorithm, it is also unnecessary for the LIS/LRI to send out a new CI, if its previous CI has not been used for the GALT calculation, that is, CISeqNum is still larger than GALTSeqNum (i.e., the sequence number of GALT calculation) (Line 8). The transient message problem, which might happen in synchronous TM algorithms, is solved using acknowledgements in a similar way to Samadis Global Virtual Time (GVT) algorithm (Samadi 1985). The variable minTSOfTransientMessages (i.e., the Minimum TS of transient messages sent by the LIS/LRI) keeps track of the minimum TS of transient messages sent by the local LIS/LRI. It is included in the CI calculation (Line 13) to ensure that the transient messages are considered in the GALT calculation.

Before CI is sent to other LIS/LRIs (Lines 10, 11), messages in the OptMsgFilter with TS $<$ min(ConReqTime, ConTSOHead)+LA are identified as incorrect messages and retractions of these messages are sent to the receivers. CON can be granted to ConLGT $=$ min(ConReqTime, ConTSOHead), if the LIS/LRI will not receive any new message with TS $<$ min(ConReqTime, ConTSOHead) after CI is sent. Otherwise, CON is granted to ConLGT $<$ min(ConReqTime, ConTSOHead). In the first case, CON will not send any message with TS $<$ min(ConReqTime, ConTSOHead)+LA in the future. Thus, the messages in the OptMsgFilter with TS $<$ min(ConReqTime, ConTSOHead)+LA are only sent by OPT and can be identified as incorrect messages. In the second case, CON may generate messages with TS $<$ min(ConReqTime, ConTSOHead)+LA. However, these messages must be generated as a result of handling the new messages received by the LIS/LRI. Obviously, these generated messages are different from those messages already in the OptMsgFilter that have TS $<$ min(ConReqTime, ConTSOHead)+LA, since the latter were generated by OPT before the new messages were received. Thus, in the second case, the messages kept in the OptMsgFilter with TS $<$ min(ConReqTime, ConTSOHead)+LA can also be identified as incorrect messages. Similarly, messages generated by OPT with TS $<$ min(ConReqTime, ConTSOHead)+LA are identified as incorrect messages and ignored simply by the LIS/LRI, if CON is in time advancement state.

Retracting the incorrect messages before sending out CI is also important to avoid delivering incorrect messages to CON replica of the receiving federate. From the perspective of the sending federate, it is obvious that the following is true for the CI it sends to the other LIS/LRIs: the value of CI $\leq$ min(ConReqTime, ConTSOHead)+LA $\leq$ OptMsgHead $\leq$ the minimal TS of all unretracted incorrect messages in the OptMsgFilter (if any). From the perspective of receiving federate, it is also obvious that GALT $\leq$ the value of CI sent by the sending federate and ConLGT $<$ GALT. By combining the above inequations, we can finally derive that the minimal TS of any unretracted incorrect messages from the sending federate must be greater than ConLGT of the receiving federate. Since the LIS/LRI only delivers CON the callbacks with TS equal to ConLGT, the unretracted incorrect messages, if any, from the sending federate will not be delivered to CON of the receiving federate.

Figure 5 shows the pseudo code of FQR, in which OptTSOHead refers to the minimum TS of messages in OptTSOQueue. Similarly, GALT is compared with OptReqTime and OptTSOHead to check whether the OPT's logical time can be advanced to min(OptReqTime, OptTSOHead) (Lines 5-7). If not, the LIS/LRI will try to update the value of GALT by sending a CI to LIS/LRIs of other federates (Lines 11-19). The minRequestRetraction (i.e., the minimal TS of TSO messages that will be removed by the Request Retraction callbacks buffered in OPT Non-message Callback Queue) must be taken into account in the CI calculation. The reason is that, the OPT might retract a previously sent TSO message with TS $\geq$ minRequestRetraction + LA after receiving the buffered Request Retraction callbacks. After the GALT has been updated, the LIS/LRI will check again whether the OPT's logical time can be advanced to min(OptReqTime, OptTSOHead). If the GALT is not updated or GALT value is still less than OptReqTime and OptTSOHead, then the OPT can only advance its logical time to GALT (Lines 16-18, 24).

Figure 6 shows the pseudo code of ReceiveCI service, which is invoked when the local LIS/LRI receives a CI form other LIS/LRIs. A new value of GALT can be calculated if the local LIS/LRI has sent out CI-seqNumOfNewCI and has received CI-seqNumOfNewCI from LIS/LRIs of all other federates (Lines 1-4). If CON is in time advancing state, the new GALT is compared with ConReqTime and ConTSOHead to check whether a TAG can be delivered (Lines 5-9). If a TAG cannot be delivered, CON remains in time advancing state and the LIS/LRI has to send a CI out for a new round of GALT calculation (Lines 12-15).

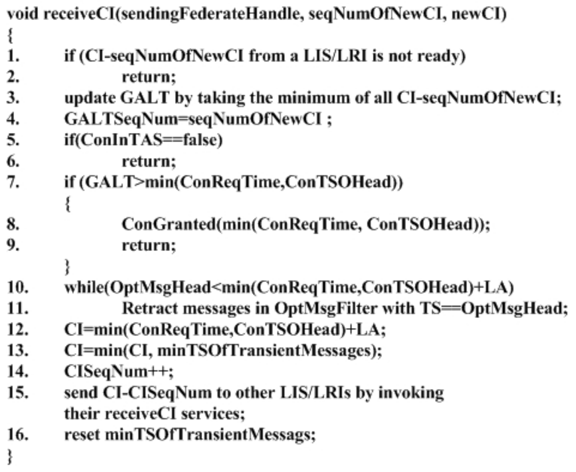
```
void FQR(t)
{
1.    if(OptInTAS==true)
2.         throw InTimeAdvancingStateException;
3.    OptInTAS=true;
4.    OptReqTime=t;
5.    if (GALT>min(OptReqTime,OptTSOHead))
      {
6.         OptGranted(min(OptReqTime,OptTSOhead));
7.         return;
      }
8.   if (ConInTAS==true||CISeqNum>GALTSeqNum)
      {
9.         OptGranted(GALT);
10.        return;
      }
11.   CI=min(OptReqTime, OptTSOHead, minRequestRetraction)+LA;
12.   CI=min(CI, minTSOfTransientMessages);
13.   CISeqNum++;
14.   send CI-CISeqNum to other LIS/LRIs by invoking
      their receiveCI services;
15.   reset minTSOfTransientMessages;
16.   if (CI-CISeqNum from a LIS/LRI is not ready)
      {
17.        OptGranted(GALT);
18.        return;
      }
19.   update GALT by taking the minimum of all CI-CISeqNum;
20.   GALTSeqNum=CISeqNum;
21.   if (GALT>min(OptReqTime,OptTSOHead))
      {
22.        OptGranted(min(OptReqTime,OptTSOhead));
23.        return;
      }
24.   OptGranted(GALT);
25.   return;
}

void OptGranted(t)
{
26.   OptLGT=t;
27.   LGT=max(ConLGT, OptLGT);
28.   reclaimMsgFilter(LGT+LA);
29.   buffer TAG(ConLGT) in Non-Message Callback Queue of OPT;
30.   OptInTAS=false;
}
```

Figure 5: Implementation details of FQR.

Similar to the implementation of NMR, the LIS/LRI tries to retract incorrect messages before sending out CI (Lines 10, 11).

## 5 EXPERIMENTS AND RESULTS

Experiments are designed to evaluate the performance of our replication structure. The federation is composed of two federates, each of which has a local event initially. When processing a local event, in addition to generates a local event, the federate may send an external event to the other federate with TS equal to TS of current event + LA. When processing an event received from other federate, the federate simply outputs some information instead of generating a local event and sending an external event. The LA of both federate is set to 5. The probability of generating an external event is denoted as $P_{ExternEvent}$. The simulation time of the generated local event is increased



```
void receiveCI(sendingFederateHandle, seqNumOfNewCI, newCI)
{
    if (CI-seqNumOfNewCI from a LIS/LRI is not ready)
        return;
    update GALT by taking the minimum of all CI-seqNumOfNewCI;
    GALTSeqNum=seqNumOfNewCI ;
    if(ConInTAS==false)
        return;
    if (GALT>min(ConReqTime,ConTSOHead))
    {
        ConGranted(min(ConReqTime, ConTSOHead));
        return;
    }
    while(OptMsgHead<min(ConReqTime,ConTSOHead)+LA)
        Retract messages in OptMsgFilter with TS==OptMsgHead;
    CI=min(ConReqTime,ConTSOHead)+LA;
    CI=min(CI, minTSOfTransientMessages);
    CISeqNum++;
    send CI-CISeqNum to other LIS/LRIs by invoking
    their receiveCI services;
    reset minTSOfTransientMessags;
}
```

Figure 6: Implementation details of ReceiveCI.

by a random number between 10 and 50. The termination time for both federates is set to 15000 simulation units. The spin-loop for processing an event is set to 30 milliseconds. To demonstrate variable performance between CON and OPT, spin-loops (10 milliseconds) are also used to emulate the cost of state saving and restoration mechanisms in OPT.

The experiments were carried out in a cluster, which has an infiniBand connection. Each of its machines is installed with 2*Dual core Xeon 3.0GHZ, 4G RAM and Redhat Enterprise 4 OS. The RTI Index service and the HLA RTI Management Services are executed on separate machines. Two LSes are deployed on different machines, each of which is connected with a CON and/or an OPT of the same federate. The CON and OPT are also executed on separate machines.

In Figure 7, we report the execution time of the simulation versus $P_{ExternEvent}$. Different synchronization approaches are considered, including a conservative approach, an optimistic approach, and the replicated approach proposed in this paper. In the conservative approach, each federate is implemented using a CON replica, and correspondingly in the optimistic approach an OPT replica is used to implement each federate. Each federate has a CON and an OPT replicas in the replicated approach.

As we can see, when $P_{ExternEvent}$ is small, the optimistic approach outperforms the conservative approach. The reason is that the federates seldom send external events to each other and only a small number of rollbacks occur in both federates. However, with an increasing $P_{ExternEvent}$, the number of rollbacks is also increased and the performance of the conservative approach becomes better than the optimistic approach.

The most important observation in this experiment is that the replicated approach outperforms both conservative and optimistic approaches. The slower replica in our replication

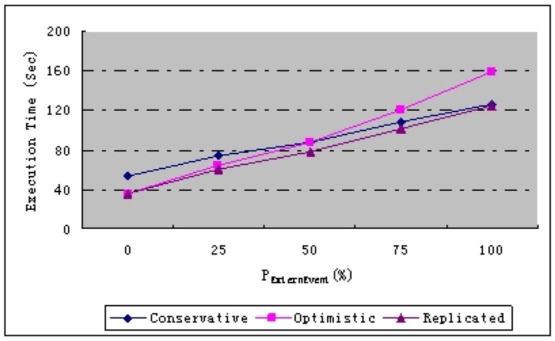


Figure 7: Execution time for different synchronization approaches with constant $P_{ExternEvent}$.

Table 2: Variable $P_{ExternEvent}$

| **Cases** | $P_{ExternEvent}$ **list** |
|---|---|
| 1 | 0%, 25%, 50%, 75%, 100% |
| 2 | 100%, 75%, 50%, 25%, 0% |
| 3 | 0%, 100%, 0%, 100%, 0% |
| 4 | 0%, 50%, 100%, 50%, 0% |
| 5 | 100%, 50%, 0%, 50%, 100% |

structure can easily catch up or even become the faster one. The request of time advancement from the slower replica can be granted directly with the current value of GALT, which has been increased by the faster replica. Experiments have also shown that CON and OPT replicas are terminated almost at the same time.

Based on the observation of the above experiment, different values of $P_{ExternEvent}$ lead to different performance for different synchronization approaches. If the value of $P_{ExternEvent}$ varies in the same execution, the optimistic approach can outperform the conservative approach during one execution phase, while the conservative approach can outperform the optimistic approach during another execution phase. We expect the replicated approach can outperform both conservative and optimistic approaches, as it always chooses the faster replica to represent the federate during the whole execution. In the second experiment, we measure the performance of the federation with variable $P_{ExternEvent}$, which is changed every 3000 simulation units. Five cases are investigated, as shown in Table 2. The execution times of different approaches are shown in Figure 8. Overall, the replicated approach can improve the execution performance by 11% to 14%, compared with the better result between conservative and optimistic approaches in which federates are not replicated.

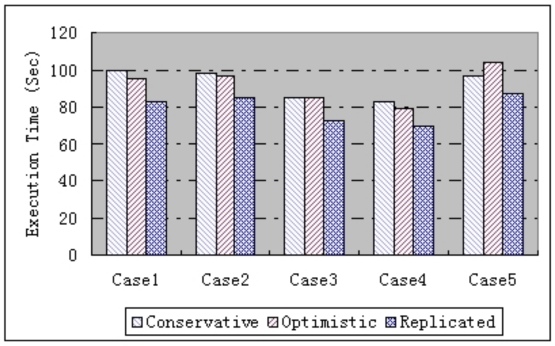


Figure 8: Execution time for different synchronization approaches with variable $P_{ExternEvent}$.

## 6 Conclusions and Future Work

In this paper we have proposed to improve the performance of a federation by replicating federates using conservative synchronization and/or optimistic synchronization approaches, and by automatically choosing the fastest replica to represent the federate in the federation during the whole execution. Besides presenting the implementation details of our replication structure based on the decoupled federate architecture in SOHR, we have also reported the experimental results to demonstrate the benefits of our approach in terms of performance improvement. In the future, we will investigate how fault tolerance can be supported by exploiting the advantages of this replication structure.

## REFERENCES

Ball, D., and S. Hoyt. 1990. The adaptive time-warp concurrency control algorithm. In *Procs of the SCS Multiconference on Distributed Simulation*, 174–177.

Bryant, R. E. 1977. Simulation of packet communication architecture computer systems. Technical report, MIT. Cambridge, MA, USA.

Chandy, K. M., and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Software Eng.* 5 (5): 440–452.

Defense Modeling and Simulation Office. *High level architecture rti 1.3ng programmers guide version 5*.

Foster, I., C. Kesselman, and S. Tuecke. 2001. The anatomy of the Grid - enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.* 15 (3): 200–222.

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*. Wiley Interscience.

IEEE 2000. *Standard 1516 (HLA rules), 1516.1 (federate interface specification) and 1516.2 (object model template)*.

Jefferson, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7 (3): 404–425.

Jha, V., and R. L. Bagrodia. 1994. A unified framework for conservative and optimistic distributed simulation. *ACM SIGSIM Simulation Digest* 24 (1): 12–19.

Niewiadomska-Szynkiewicz, E., and A. Sikora. 2002. Algorithms for distributed simulation - comparative study. In *Procs of International Conference on Parallel Computing in Electrical Engineering*, 261–266.

Pan, K., S. J. Turner, W. Cai, and Z. Li. 2007. A service oriented HLA RTI on the Grid. In *Procs of International Conference on Web Services*, 984–992.

Pan, K., S. J. Turner, W. Cai, and Z. Li. 2008. A hybrid HLA time management algorithm based on both conditional and unconditional information. In *Procs of 22th Workshop on Parallel and Distributed Simulation*, 203–211.

Quaglia, F. 2006. A middleware level active replication manager for high performance HLA-based simulations on smp systems. In *Procs of 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 219–226.

Rajaei, H., R. Ayani, and L.-E. Thorelli. 1993. The local Time Warp approach to parallel simulation. *ACM SIGSIM Simulation Digest* 23 (1): 119–126.

Samadi, B. 1985. *Distributed simulation, algorithms and performance analysis*. Ph. D. thesis, University of California, Los Angeles.

Sokol, L. M., D. P. Briscoe, and A. P. Wieland. 1988. MTW: A strategy for scheduling discrete simulation events for concurrent execution. In *Procs of the SCS Multiconference on Distributed Simulation*, 34–42.

Xu, S., and L. F. McGinnis. 2006. Optimistic-conservative synchronization in distributed factory simulation. In *Procs of the 2006 Winter Simulation Conference*, 1069–1074.

## AUTHOR BIOGRAPHIES

**ZENGXIANG LI** is a PhD student in the School of Computer Engineering at Nanyang Technological University (NTU) in Singapore. He received his M.Eng. from the Shanghai Jiao Tong University (China).

**WENTONG CAI** is an Associate Professor in the School of Computer Engineering at NTU. He received his PhD in Computer Science from University of Exeter (UK).

**STEPHEN JOHN TURNER** is a Professor in the School of Computer Engineering and Director of PDCC at NTU. He received his PhD in Computer Science from Manchester University (UK).

**KE PAN** is a PhD student in the School of Computer Engineering at NTU. He received his B.Eng. from NTU.