Transparent Optimistic Synchronization in the High-Level Architecture via Time-Management Conversion

ANDREA SANTORO, Banca d'Italia
FRANCESCO QUAGLIA, Sapienza Università di Roma

Distributed simulation allows the treatment of large/complex models by having several interacting simulators running concurrently, each one in charge of a portion of the model. In order to effectively manage integration and interoperability aspects, the standard known as High Level Architecture (HLA) has been developed, which is based on a middleware component known as Run-Time-Infrastructure (RTI). One of the main issues faced by such a standard is synchronization, so that HLA supports both conservative and optimistic approaches. However, technical issues, combined with some peculiarities of the optimistic approach, force most simulators to use the conservative approach. In order to tackle these issues, we present the design and implementation of a Time Management Converter (TiMaC) for HLA based simulation systems. TiMaC is a state machine designed to be transparently interposed between the application layer and the underlying RTI, which performs mapping of the conservative HLA synchronization interface onto the optimistic one. Such a mapping allows transparent optimistic execution (and the related benefits) for simulators originally designed to rely on conservative synchronization. This is achieved without the need to modify the RTI services or alter the HLA standard. An experimental evaluation demonstrating the viability and effectiveness of our proposal is also reported, by integrating our TiMaC implementation with the Georgia Tech B-RTI package and running on it both (A) benchmarks relying on traces from simulated demonstration exercises collected using the Joint Semi-Automated Forces (JSAF) simulation program and (B) a self-federated Personal Communication System simulation application.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event*; *parallel*

General Terms: Theory, Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Optimistic synchronization

ACM Reference Format:
Santoro, A. and Quaglia, F. 2012. Transperent optimistic synchronization in the high level architecture via time-management conversion. ACM Trans. Model. Comput. Simul. 22, 4, Article 21 (November 2012), 26 pages.
DOI = 10.1145/2379810.2379814 http://doi.acm.org/10.1145/2379810.2379814

1. INTRODUCTION

The High Level Architecture (HLA) is a standard for the integration and the interoperability of autonomous simulators [IEEE 2000a]. Its target is the building of complex simulation systems (*federations* in the HLA terminology) through the adoption of a Run-Time-Infrastructure (RTI) acting as a middleware component, which offers a general set of services to each involved simulator (i.e., to each *federate*). Federates

Authors' addresses: A. Santoro, Banca d'Italia, Via Otricoli 41, 00181 Roma, Italy; email: andrea.santoro@mclink.it; F. Quaglia, Dipartimento di Ingegneria Informatica, Automatica e Gestionale "Antonio Ruberti", Sapienza Università di Roma, Via Ariosto 25, 00185 Roma, Italy; email: quaglia@dis.uniroma1.it.

DOI 10.1145/2379810.2379814 http://doi.acm.org/10.1145/2379810.2379814

are allowed to run concurrently on different CPUs/cores so that the computing power offered by parallel or distributed platforms can be exploited while supporting the execution of the whole simulation model.

The HLA interface specification [IEEE 2000b] defines a suite of services, called Time Management, to be offered by the RTI in support of synchronized execution among concurrent federates. However, while it is relatively easy to implement the conservative synchronization approach using the HLA, the optimistic approach introduces a set of additional complications. One complication is related to the fact that this kind of synchronization requires checkpointing and state recovery mechanisms to be built within the federate-level code (these mechanisms are not in charge of the RTI component). Another complication is the burden of handling at the application level the whole set of tasks related to rollback management (e.g., event retractions), and also their schedule.

To provide a building block to address such an issue, Santoro and Quaglia [2005a, 2005b] presented a software architecture, called MAgic State Manager (MASM), allowing transparent checkpointing/recovery of the federate state. In this article we present the design and implementation of a Time-Management-Converter (TiMaC) that, exploiting the capabilities of MASM, allows completely transparent optimistic synchronization. TiMaC is a state machine that can be transparently interposed between the federate-level software and a conventional RTI. It exposes to the overlying federate the Time Management interface proper of conservative synchronization, while actually supporting optimistic synchronization (hence interacting with the RTI via optimistic synchronization services). In other words, TiMaC performs a runtime mapping between the (conservative) synchronization mode seen by the overlying federate, and the real (optimistic) synchronization mode selected for the execution. The whole design has been based on a classification of HLA services which we introduce in this article, and on the definition of adequate policies for service treatment while supporting the mapping of conservative onto optimistic Time Management modes. These policies can be also used to cope with services offered to the federate by (third party) programming libraries and/or by the Operating System kernel (e.g., I/O services), which need adequate treatment while supporting transparent optimistic execution of the federate-level software.

The implementation of TiMaC we have developed has been tailored for integration with the Georgia Tech B-RTI package [Fujimoto et al. 2000], namely a `C`-based HLA-compliant release of the RTI middleware component, even though the design principles used for developing the implementation remain valid independently of the specific underlying RTI. In order to quantify the effectiveness of our proposal, we also report the results of an experimental study based on both (A) synthetic benchmarks derived by exploiting traces collected from simulated demonstration exercises relying on the Joint Semi-Automated Forces (JSAF) simulation program and (B) a self-federated Personal Communication System (PCS) simulation application.

This article significantly expands early results presented by Santoro and Quaglia [2006]. First, the systematic treatment of the complete suite of HLA services provided in this article was not present within the original TiMaC proposal, which was therefore limited to the handling of a restricted set of message-exchange services. Second, the experimental assessment in the conference article was limited to self-federation of three instances of the PCS simulator. Instead, in this article we provide a wider study relying on a suite of trace-based benchmarks, where a much larger number of parameters are varied (including differentiated policies for federate-deploy over the computing platform), and we also consider scaled-up federation size, also for the case of the PCS application.

The remainder of this work is structured as follows. Related work is discussed in Section 2. In Section 3 a brief overview of MASM is presented. In Section 4 we provide

a recall on Time Management services. The design of TiMaC is presented in Section 5. The experimental study is provided in Section 6.

2. RELATED WORK

Issues and complications related to optimistic synchronization in HLA have been addressed by Vardanega and Maziero [2000] and by Wang et al. [2005] via the introduction of rollback managers (or controllers). The target of these proposals is to mask some interactions (e.g., some RTI callbacks) so to avoid the need for having the corresponding (callback) management modules within the application-level code. In other words, they show to the overlying federate a simplified interface for optimistic synchronization, although they rely on the use of two special callbacks towards the federate code, namely *GetState* and *SetState*, which are even not prescribed by the HLA specification. Compared to these proposals, our approach is different in nature since its aim is to support a complete, transparent conversion of the conservative interface into the optimistic one, performed by TiMaC with no need for any module in support of optimistic synchronization within the application level code.

Cai et al. [2005] provide message-treatment rules for causally-ordered delivery of messages across federates [Fujimoto and Weatherly 1996], which represents an intermediate delivery semantic with respect to those encompassed by HLA-native Time Management [IEEE 2000b], namely timestamp-ordering and receive-ordering (see Section 4 for an overview). This approach has been shown to be viable when actuated on-top of conventional Time Management services. Hence, as a matter of fact, it is orthogonal to our proposal, since, in principle, it could be still layered on top of the TiMaC state machine, which exposes to the overlying software the same identical interface offered by an HLA-compliant RTI.

Our work is also related to the proposal by Chen et al. [2005], where an intermediate layer is interposed between the application and the RTI in order to support treatment logics of HLA call/callback services aimed at providing enhanced execution modes. However, differently from our approach, this work is aimed at optimizing the exploration of differentiated simulation scenarios via computation reuse, which is achieved by the introduction of HLA-oriented cloning mechanisms. Instead, as hinted, our aim is to optimize model execution on each single run (or scenario) by transparently enabling optimistic synchronization, hence speculative processing. As a matter of fact, the two approaches can be again considered as orthogonal to each other.

In the context of traditional (non HLA-based) parallel/distributed simulation, several proposals have addressed issues related to the tradeoff between conservative and optimistic synchronization. This has been done via frameworks and/or architectures aimed at jointly supporting both types of synchronization by either entailing the possibility of dynamically switching between one and the other synchronization mode [Bagrodia et al. 1991; Jha and Bagrodia 1994; Steinman 1992; Perumalla 2005], or by having the two synchronization modes employed in a hierarchical fashion [Rajaei et al. 1993]. Also, some other works have been focused on dynamically controlling the actual level of optimism [Reynolds, Jr. 1988; Srinivasan and Reynolds, Jr. 1998; Das and Fujimoto 1997]. Differently from all these results, our proposal is not targeted at defining innovative mixtures of conservative and optimistic modes. Instead, we rely on an orthogonal approach where an ad-hoc protocol stack based on the introduction of the TiMaC layer is used exclusively for decoupling the actual synchronization mode operated by a federate, as seen by the RTI, from the Time Management interface seen by the federate. Overall, with such a protocol stack, the actual distributed synchronization is anyhow demanded to the RTI, which already supports both conservative and optimistic modes, and the target is transparency in the exploitation of the optimistic mode. This necessarily requires proper management schemes, which have been embedded

within the TiMaC state machine. Again on the side of synchronization transparency, the innovation by TiMaC is that it copes with a set of application-exposed HLA services that is relatively wider compared to the counterpart offered to the application layer in the context of traditional parallel/distributed simulation systems [Perumalla 2005].

3. MASM OVERVIEW

Although being defined as separate objects by the HLA standard, the federate and the underlying simulation middleware, that is, the RTI, are typically parts of a same application program, thus they are executed within the same process and share the same data area. Additionally, classical RTI process models, supported by many commercial and noncommercial implementations [DMSO 2004; Georgia Tech Research Corporation 2003; Virtual Technology Corporation 2004], are based on interactions between the federate and the RTI through methods exposed by RTI and invoked by the federate. Hence, the federate and (at least a portion of) the RTI are executed within the same thread, thus they share the same stack. These peculiarities make techniques traditionally employed to checkpoint/recover a whole process state transparently, like the one by Plank et al. [1995], not directly applicable to the HLA context, where we have a single process/thread and we want to transparently checkpoint/recover only a portion of its whole state selectively. To solve this problem, MASM adopts a set of compile-time and runtime techniques allowing the split of the whole process memory areas into two separate sections. One section contains the RTI state, and the other one contains the federate state. A checkpoint operation performed by MASM will save the content of the federate section into buffers within the RTI section. Conversely, a recovery of the federate state will consist in copying the saved data back into the federate section. The compile-time techniques deal with such a separation for what concerns statically allocated memory, whereas, the runtime techniques deal with such a separation for what concerns both the heap and the stack areas.

MASM performs checkpointing by incrementally logging only dirty pages within the memory image portion related to the federate. This is achieved by exploiting Operating System memory protection mechanisms. Although a finer-grain approach to incremental checkpointing could further reduce the overhead, the page-based approach exhibits the advantage of application semantic independence (hence allowing checkpoints of the federate state even in case it is scattered on dynamically allocated memory chunks), while allowing for high portability [Santoro and Quaglia 2005b]. This is not supported by finer-grain approaches [Ronngren et al. 1996; Steinman 1993; West and Panesar 1996], which require explicit declaration of which portions of the address space need to be part of a snapshot. On the other hand, solutions like the one by Pellegrini et al. [2009] support full transparency of fine-grain incremental logging, even for state layouts based on dynamically allocated memory areas, but exhibit reduced portability due to the employment of architecture-dependent instrumentation schemes for tracing memory writes. MASM API entails the following.

—`SaveState(Timetype VirtualTime)`. This service performs an incremental checkpoint of the federate state. The input parameter is the current simulation time for the federate, which is associated by MASM with the currently taken checkpoint.
—`RecoverState(Timetype VirtualTime)`. This service restores a previously checkpointed state. The input parameter is the simulation time to be recovered for the federate, which determines to checkpointed state selected for the recovery procedure.
—`PruneStateLog(Timetype CommittedTime)`. This service frees the memory that keeps the oldest checkpoints. All checkpoints whose simulation time is lower than the value of `CommittedTime` are removed from the log.

The design of TiMaC we present is independent of the specific MASM version since it does not require interactions with MASM internals. Hence, alternative MASM releases could be employed [Santoro and Quaglia 2005a, 2005b] exhibiting different performance vs portability tradeoffs.

4. RECALL ON TIME MANAGEMENT SERVICES

In this section we recall relevant features characterizing both the conservative and the optimistic Time Management interfaces. Our focus is on the main differences between these interfaces, which are essentially related to the different handling of the two categories of messages associated with interaction and coordination along the HLA time-axis, namely TimeStamp Ordered (TSO) and Receive Ordered (RO) messages [IEEE 2000a]. Such a different handling only deals with the treatment of incoming messages (hence not concerning messages possibly produced in output by the federate). This is natural since synchronization deals with the advancement in simulation time subject to (possible) delivery of incoming TSO messages.

As a last preliminary observation, we remark that HLA is based on the concepts of time-regulation and time-constrainment. A time-regulating federate is allowed to insert in the system messages that can obey the TSO rule concerning the delivery at the destination side. A time-constrained federate must process TSO messages according to the order specified by timestamps. This does not exclude the possibility for a TSO message sent by a time-regulating federate to be treated at destination like an RO message (and hence to be delivered according to the order in which it is received by the destination RTI instance) due to the fact that the destination federate is not time-constrained. Similarly, for a TSO message sent by a federate which is not time-regulating, its actual treatment is to deliver that message at the destination side as an RO message. Concerning this point, each time we mention the concept of outgoing/incoming TSO (or RO) message we implicitly mean the actual treatment of the message depending on whether the sender/recipient federate is time-regulating/time-constrained or not. The definition of actual message treatment depending on time-regulation and time-constrainment can be found in [IEEE 2000b], Section 8.1.1.

4.1. Conservative Synchronization

The typical RTI service used for conservative synchronization of event-driven federates is *Next Message Request* (NMR)[1]. This service allows the federate to ask for an advancement of its local simulation clock to a given value t specified as an input parameter to the service invocation (typically this is the time of the next event in the federate local event queue). The RTI grants to the federate the advancement to simulation time t only in case there is no possibility for a TSO message to be delivered to that federate in the future with timestamp less than or equal to t. In other words, time t is safe for the federate advancement. (Such a safe time is known as LBTS or GALT.) Otherwise the granted time will be less than t, and will depend on the minimum possible timestamp of incoming messages for that federate. The grant is delivered to the federate through the *Time Advance Grant* (TAG) callback invoked by the RTI. The RTI also performs the following actions before granting the federate via TAG: (i) It delivers to the federate all the TSO messages with timestamp equal to the time of the grant, if any; (ii) It delivers to the federate all the incoming RO messages already buffered by the RTI, if any. Given that no TSO message with timestamp less than or equal to the time of the

[1]A minor variation of this service is called *Next Message Request Available*, which only differs from NMR in the federate guarantee about the minimum timestamp of future outgoing TSO messages. Also, for time-stepped federates, there are Time Management services analogous to NMR and NMRA, which are called *Time Advance Request* (TAR) and *Time Advance Request Available* (TARA).

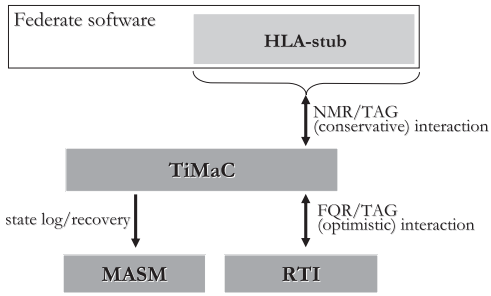

Fig. 1. System architecture with TiMaC.

grant will be ever delivered to the federate in the future, all the messages delivered to the federate via the conservative Time Management interface are guaranteed to be causally consistent along simulation time.

4.2. Optimistic Synchronization

For optimistic synchronization, the typical Time Management service is called *Flush Queue Request* (FQR). With this service the federate can ask the RTI to be granted to a given simulation time t passed as input parameter to FQR. Also in this case the RTI replies to a FQR call with a TAG callback, indicating which is the real value of the safe simulation time for the federate (i.e., which is the real value of GALT). This value might be less than or equal to t. The following actions are executed by the RTI before delivering the TAG in reply to an FQR invocation: (i) It delivers to the federate all the already buffered RO messages; (ii) It delivers to the federate all the already buffered TSO messages, independently of their timestamp value (which might be therefore larger than the time of the grant). By point (ii), TSO messages might arrive in the future with timestamps less than the timestamp of the messages that are delivered by the RTI prior to providing the grant to the federate. Hence, the optimistic Time Management services can cause the delivery of messages independently of whether these messages are causally consistent in simulation time.

4.3. Message Delivery Supports

For both conservative and optimistic synchronization, the delivery of TSO and RO messages to the federate, after an NMR or FQR request is issued, occurs through a set of callbacks that RTI can invoke. These belong to the class of so called Object Management services specified by HLA. The typical callbacks are *Reflect Attribute Values* (RAV), *Receive Interaction* (RI) and *Remove Object Instance* (ROI). As said, the delivery of messages, and hence these callbacks, are issued by the RTI after the NMR or FQR request from the federate and before the TAG callback. The typical trigger for all these callbacks (including TAG) is an additional call to the RTI-tick service (also defined as Evoke Callback) issued by the federate after NMR or FQR.

5. DESIGN OF THE TIME MANAGEMENT CONVERTER

5.1. Design Concepts

The TiMaC layer implements a state machine interposed between the federate and the RTI (see Figure 1), which interacts with the other layers as follows.

Upper Level Interaction (ULI). TiMaC interacts with the overlying federate-level code via the conventional conservative interface based on NMR and TAG. This

is done in order to provide the federate with the illusion to proceed according to conservative synchronization.
Lower Level Interaction (LLI). TiMaC interacts with the underlying RTI via the conventional optimistic interface based on FQR and TAG. This is done in order to provide the RTI with the illusion that the federate-level software is executing according to optimistic synchronization.

Different instances of TiMaC do not directly interact with each other. This is a natural choice since TiMaC is not intended to reflect the same (or similar) functionalities offered by the standard RTI, which is in charge of executing distributed protocols across its instances. Based on this view, the role of ULI and LLI is as follows.

—TiMaC exploits LLI both (i) to detect a *globally safe* simulation time for local advancement (i.e., the GALT advancement within the federation), which occurs via the TAG callback from the RTI, and (ii) to enlarge its local knowledge about messages already injected within the system and destined to the overlying application. Recall, in fact, that when FQR is invoked over LLI, then the RTI delivers to TiMaC any already buffered message destined to the overlying application, independently of whether the message is causally consistent.
—TiMaC exploits ULI to notify to the overlying federate both (i) a *locally safe* simulation time for the federate advancement (this is not necessarily compliant with the real GALT advancement), which occurs via the TAG callback from TiMaC, and (ii) messages (e.g., TSO messages) which are locally detected as causally consistent (the delivery of the messages occurs via RAV, RI and ROI callbacks, this time issued by TiMaC). Both the locally safe simulation time in point (i) and the messages in point (ii) are determined on the basis of TiMaC local (partial) knowledge about the whole system state. As an example, TiMaC can identify the minimum timestamp value of some TSO message it currently keeps buffered (which has been previously received from the RTI via LLI) as the time of the grant. On the basis of TiMaC local (partial) knowledge this message is causally consistent since no other TSO message destined to the federate, which TiMaC is aware of, has a lower timestamp value.

In other words, the objective of such a decoupling between ULI and LLI is to exploit LLI to receive the largest amount of information already available at the underlying RTI concerning incoming messages, in order to increase the likelihood that the interactions with the federate-level software over ULI (e.g., the delivery of a TSO message with timetamp t and the associated TAG towards the federate) do not eventually reveal inconsistent (e.g., due to future delivery of an incoming TSO message over LLI with timestamp less than t). In case some message delivery and/or grant occurring over ULI is eventually detected as inconsistent, TiMaC can recover the effects of such an interaction on the federate state via the facilities offered by MASM. In fact, as shown in Figure 1, TiMaC also directly interacts with MASM, thus having the possibility to log and recover virtual memory pages associated with federate-level data structures.

However, the whole set of interactions allowed by HLA over both conservative and optimistic Time Management interfaces are not exclusively limited to the delivery of (RO and TSO) messages and grants for simulation time advancement issued by the RTI. Instead, they include other kinds of interactions possibly issued by the federate (e.g., service calls towards the RTI), and other kinds of callbacks, different from TAG and from the ones used for message delivery operations (i.e. RAV, RI and ROI). As an example, no message would be ever delivered to any federate from the RTI if that same message were not inserted in the system by some federate, which occurs via a set of other services belonging to the Object Management class. Concerning this issue, common services for sending messages are *Update Attribute Values* (UAV), *Send*

Interaction (SI) and *Delete Object Instance* (DOI), which will result in the previously described RAV, RI and ROI callbacks delivering the corresponding information at the destination side. More in general, calls/callbacks associated with differentiated services specified by HLA can occur while Time Management takes place over both the conservative and the optimistic interfaces. Hence, we need to define a correct treatment of whichever call/callback to be performed by TiMaC while mapping the NMR/TAG conservative interface over the FQR/TAG optimistic interface, where "correct" means that any recovery action performed by TiMaC due to the tracking of a causally inconsistent execution over ULI must always ensure mutual consistency of the states of the federate-level software and the underlying RTI software. This is not trivial since these states are (possibly) mutually affected due to both calls and callbacks issued by each layer.

To provide a correct treatment we rely on a classification of HLA services (both calls and callbacks) based on the notion of idempotence and observability as specified in the following section. Then we introduce a set of policies allowing adequate treatment of services within each class, which are used to support the actual Time Management mapping performed by TiMaC. NMR, FQR and TAG are excluded from this classification since they represent special services whose mapping over each other (across ULI and LLI) exactly defines the problem we address via the service classification and the introduction of the service management policies. With no loss of generality we also exclude from the classification the RTI-tick service, since it is an auxiliary service used in some implementations to simply pass the control from the federate to the RTI, hence only aiming at defining the wall-clock-time frames where callbacks can occur.

5.2. Classification: Idempotence and Observability of HLA Services

A first differentiation among general HLA services (both calls and callbacks) can be based on whether the effects of the execution of a given service can be explicitly undone via the invocation of another service or not. As an example, some HLA services are associated with retraction designators which can be used in order to invoke a different service to retract the original service invocation. On the other hand, other HLA services are associated with no retraction designator. Hence, once issued by whichever layer (federate or RTI), these services have no way to be explicitly undone. As an example, concerning Object Management services, there exists a service that can be used for retracting a previously sent message, namely the *Retract* (R) service. This might result in the *Request Retraction* (RR) callback on a remote federate in case the delivery has been already executed by the destination RTI. However, a retraction can be invoked only in case the corresponding sent message is a TSO message (in this case the send returns a federation-unique retraction designator). Instead, no retraction can be invoked in case of a sent RO message.

By the previous observations, the possibility (or the impossibility) to explicitly retract the effects of a service call needs to be considered when performing the mapping of conservative to optimistic Time Management. By abstracting over the concepts of the "effects" of a given RTI service, we can envisage the following property.

Idempotence Property ($\mathcal{I}$). We say that an HLA service (call or callback) is idempotent if and only if: ($\mathcal{I}$.1) Its execution does not alter the internal state of the RTI; or ($\mathcal{I}$.2) It can be retracted. Otherwise we say that the RTI service is nonidempotent.

Concerning point $\mathcal{I}$.1, we note that each HLA service is specified also in terms of so called postconditions defining whether the RTI (or the federate) state is affected by the service execution. Hence by those postconditions we can identify services which, although nonretractable, are idempotent due to point $\mathcal{I}$.1.

According to the $\mathcal{I}$ property, UAV, SI and DOI calls from the federate associated with TSO messages represent idempotent HLA services, since they can be retracted (i.e., they are idempotent due to $\mathcal{I}.2$). Instead, UAV, SI and DOI calls associated with RO messages are nonidempotent since they alter the internal state of the RTI (in fact the postconditions assert that the new instance attribute values have been supplied to the RTI) but they cannot be retracted (since, according to the HLA specification, no retraction designator exists for services acting on RO messages). On the other hand, looking at the HLA specification, it is possible to find several services which, according to our specification of Property $\mathcal{I}$, are idempotent due to $\mathcal{I}.1$. Among them, we can mention several query services which can be used by the federate to acquire information on its current state within the federation, as seen by the RTI. Examples of these services are *Query GALT* or *Query Logical Time*, which belong to the Time Management class.

It is important to observe that HLA callback services are all idempotent according to Property $\mathcal{I}$. This is because, according to postcondition specification, the effect of the execution of whichever callback is to (possibly) change the internal state of the federate, not of the underlying RTI. Hence $\mathcal{I}.1$ does always hold for any callback.

Another point to be considered is whether the service returns a result or not, since TiMaC cannot provide a result for an RTI service invoked by the federate before the RTI has really executed the service and has returned that result to TiMaC. Specifically, the result may depend on the internal state of the RTI. Concerning this point, we can envisage the following property further refining the classification of HLA services.

Observability Property ($\mathcal{O}$). We say that an HLA service (call or callback) is observable if and only if it returns a result. Otherwise the service in nonobservable.

Given that HLA explicitly defines whether services have "returned arguments" or not, the binding of observability to HLA services is immediately obtained via the presence or not of the returned-arguments field within their specification. According to the $\mathcal{O}$ property, UAV, SI and DOI calls from the federate associated with RO messages represent nonobservable RTI services, since they return no value (e.g., no retraction designator). Instead, UAV, SI and DOI calls associated with TSO messages are observable since they return a retraction designator. On the other hand, looking at the previously mentioned HLA query services (e.g., *Query GALT* or *Query Logical Time*), they are all observable since they all return a result. It is important to note that callback services prescribed by the HLA specification are all nonobservable services since, according to HLA specification, no callback returns any result value to the RTI.

5.3. Policies for the Treatment of Federate Calls over ULI

We define as an "epoch" for the federate, the simulation time interval between two subsequent TAG callbacks issued by TiMaC towards the federate over ULI. By the HLA specification, a TAG towards the federate can be eventually issued by TiMaC only after the federate has issued an NMR request for advancing its local simulation clock according to the conservative Time Management interface. Hence, considering wall-clock-time, TiMaC detects that a new epoch needs to be triggered for the federate when the NMR call is issued. Given that a TAG over ULI is not necessarily compliant with the real advancement of the GALT within the federation, each epoch represents a simulation time advancement which can ultimately be either committed or rolled back. This depends on whether (future) incoming TSO messages over LLI reveal or not causality inconsistencies for the delivery of the TAG determining the upper limit of the epoch. We say that an epoch gets committed when the TAG from the underlying RTI delivered via LLI notifies to TiMaC that the GALT for the federate is greater than or

Table I. Treatment of Federate Calls to RTI Services

Type of Service	observable	nonobservable
idempotent	Immediate-Forward	Immediate-Forward
nonidempotent	Wait-Until-Commit	Delay-Until-Commit

equal to the epoch upper limit. We define the following three policies for the treatment of calls issued by the federate over ULI during any epoch.

Immediate-Forward Policy. Upon the receipt of a service call from the overlying federate during any epoch, TiMaC forwards that same service call to the underlying RTI, regardless of whether that epoch gets eventually committed or rolled back. Afterwards TiMaC returns to the federate.

Delay-Until-Commit Policy. When a service call is received from the overlying federate during any epoch, TiMaC immediately returns to the federate (as if the service call was executed), but keeps the service call pending, i.e., it does not forward it to RTI, until that epoch gets committed. In case that epoch gets eventually rolled back, the pending service call is discarded.

Wait-Until-Commit Policy. Upon the receipt of a service call from the overlying federate during any epoch, TiMaC waits the commitment/rollback of the epoch. In case the epoch is eventually committed, TiMaC forwards that same service call to the underlying RTI. Afterwards it returns to the federate. In case that epoch is eventually rolled back, the service call is discarded (i.e., not executed at all).

By using the previous policies, we list in Table I the treatment performed by TiMaC for what concerns the calls to RTI services issued by the overlying federate during any epoch. In case the invoked service is idempotent (recall that UAV, SI and DOI services for sending TSO messages are exactly within this class), then TiMaC adopts the Immediate-Forward policy independently of whether the service is observable or nonobservable. In fact, for whichever idempotent service call immediately forwarded to the RTI during an epoch that is eventually rolled back, either TiMaC can recover any effects of that service on the RTI state during the epoch rollback phase (by retracting the service) or that service call had no effect at all on the state of the RTI. The situation is different for nonidempotent RTI services since, once forwarded to the RTI, their effects cannot be eventually undone in case the corresponding epoch is eventually rolled back. For these services, we need to commit the corresponding epoch before they can be issued by TiMaC towards the underlying RTI.

Delay-Until-Commit and Wait-Until-Commit both forward a nonidempotent service call towards the underlying RTI only after the corresponding epoch gets committed (i.e., when TiMaC receives the TAG from the underlying RTI asserting that the safe simulation time for the federate is beyond the upper limit of that epoch). However, according to the two different specifications for these policies, we need to differentiate service treatment on the basis of whether a service, beyond being nonidempotent, is observable or nonobservable. In case the service is observable, TiMaC needs to provide a return value to the federate, which must be provided to TiMaC by the underlying RTI, hence we cannot return control to the federate prior to the invocation of that service towards the RTI. This is expressed by the Wait-Until-Commit policy. Otherwise TiMaC can return control to the federate without the need for waiting the service return from the RTI, which is the behavior expressed by the Delay-Until-Commit policy.

Table II. Treatment of RTI Callbacks

All services (except *Request Retraction* - RR)	RR service
Log-And-Forward	Filter

5.4. Policies for the Treatment of RTI Callbacks over LLI

The RTI can issue callbacks towards TiMaC over LLI, and this can occur when the overlying federate is executing in any epoch. Hence, we need adequate policies for the treatment of such callbacks by TiMaC, which we define in what follows.

Log-And-Forward Policy. Upon the receipt of a callback from the underlying RTI, TiMaC logs the callback information (i.e. name and parameters). If not eventually retracted by the underlying RTI, the callback will be eventually issued to the federate during some epoch. After the log operation, TiMaC returns to the RTI.

Filter Policy. Upon the receipt of a callback from the underlying RTI, TiMaC processes the callback locally, without eventually delivering it to the federate. After the processing, TiMaC returns to the RTI.

By using the previous policies, we list in Table II the treatment performed by TiMaC for what concerns the callbacks from the RTI during whichever epoch. This time the differentiation is not done on the basis of the $\mathcal{I}$ and $\mathcal{O}$ properties since, as discussed in Section 5.2, all the callbacks specified by HLA are both idempotent and nonobservable. Instead, the different treatment expressed in Table II is related to whether a given callback coming from the RTI via LLI is ever expected to occur over ULI. As shown in the table, the only callback possibly coming from the RTI via LLI, which is never expected to occur over ULI, is RR. This callback is used to retract some TSO message previously delivered by the RTI to the overlying software, which eventually reveals not to be causally consistent. Given that a federate interacts with TiMaC via the conservative Time Management interface based on NMR/TAG, it assumes that messages delivered by the underlying software are always causally consistent. This point of view must be respected by the logic embedded within TiMaC. Therefore, any incoming RR callback must be processed by TiMaC in a totally transparent manner to the federate, which is reflected by the Filter policy. Specifically, the processing of the RR callback coming from the RTI via LLI might result in a recovery operation of the federate state transparently supported by TiMaC via the facilities offered by MASM. This occurs in case the RR callback is associated with a causally inconsistent TSO message that TiMaC has delivered to the federate via ULI.

All the other callbacks (different from RR) occurring over LLI are expected to possibly occur also over ULI. Some of them might be eventually retracted by the RTI since they are subject to causality constraints. These are the callbacks associated with the delivery of TSO messages. Conversely, the other callbacks will not eventually be retracted, since they are not subject to causality constraints related to the federate advancement in simulation time. As an example, the delivery of a RO message via, for instance, the RAV callback, is not subject to future retraction by the RTI.

All the callbacks that are not eventually retracted by the RTI need to result as eventually issued towards the federate. Hence, the need arises to eventually forward these callbacks towards the overlying federate when received by TiMaC over LLI. On the other hand, transparent state recovery supported by TiMaC may affect any callback different from RR, which has been already forwarded to the federate during any epoch, and which will not eventually be retracted by the underlying RTI. In particular, the transparent state recovery procedure undoes all the updates occurring within the federate state, which are associated with the processing of the callbacks occurred during the rolled back epochs. As an example, all the RO messages delivered to the federate

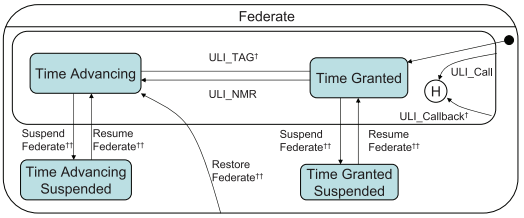

Fig. 2. Federate state-diagram.

via RAV, RI and ROI callbacks forwarded from LLI towards ULI, will result as undelivered after the state recovery operation. Hence they will result in callbacks executed by TiMaC, but no more registered as executed by the federate level software. To avoid this problem, TiMaC logs the information associated with whichever callback, so that the callback can be eventually replayed (i.e., redelivered via ULI) in case transparent state recovery undoes its effects on the federate state. This is the behavior expressed by the Log-And-Forward policy. We also note that, being all the callbacks specified by HLA both idempotent and nonobservable, they can be replayed with no side effect on the state of the RTI, since they do not touch RTI data structures and do not produce any (nondeterministic) output value towards the RTI itself.

We note that this callback-undo effect does not appear in the context of traditional optimistic federates since they are programmed aware of whether some callback related action needs to figure as undone (or not) in case some causality violation for the receipt of TSO messages occurs. Instead, TiMaC performs state recovery through MASM, which acts on the whole memory image portion associated with the federate state. This prevents the possibility to design the federate-level code in such a way to allow the federate to consciously and selectively maintain specific state updates occurring during some rolled back epoch. This is aligned with the spirit of TiMaC, which aims at fully masking state recoverability at the application level.

5.5. Treatment of Services Offered by (Standard) Programming Libraries and by the Kernel

In general settings, a federate might invoke functions offered by (standard) third party programming libraries. Even though the internal state of a library can be marked by MASM as belonging to the federate memory area, which is subject to checkpoint and recovery actions, this does not mean we can really recover any action performed by a call to a function within the library since it might have an interaction with the operating system kernel, whose actions cannot be assumed to be always recoverable. As an example, it is not possible to recover an I/O operation on the terminal (this is the well known problem of the output commit in rollback recovery systems [Elnozahy et al. 2002]). Additionally, the library might be shared across federate-level and RTI-level software. Similar considerations apply to the case of federates directly interacting with the Operating System kernel. To cope with this issue, TiMaC should adopt the Wait-Until-Commit policy for the management of federate invocations to all the standard library functions and/or system calls causing nonrecoverable actions and returning a result. On the other hand, the Delay-Until-Commit policy should be adopted in case no result is returned (but the actions are anyway nonrecoverable). This can be practically achieved by letting TiMaC intercept the calls via, for instances, wrapping approaches.

5.6. Putting Things All Together: Federate and TiMaC State-Diagrams

The preceding discussion and outcomes led to the definition of the state-diagrams in Figure 2 and in Figure 3 as representative of TiMaC based executions, where, for easiness of presentation, we focus on the HLA suite, thus not explicitly depicting the

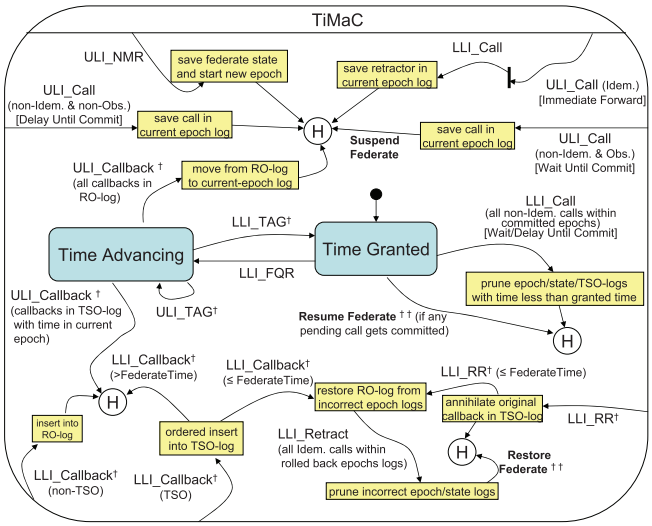

Fig. 3. TiMaC state-diagram.

management of the interactions with standard libraries and systems calls. The state-diagrams are defined according to the notation used in the HLA standards, with some additional notation that takes into account the peculiarities of TiMaC.

The notation ULI_Call (resp. ULI_Callback) is used to refer to any specific service call (resp. callback) invoked over ULI. Analogously, LLI_Call (resp. LLI_Callback) is used to refer to calls (callbacks) invoked over LLI. Sometimes, the call name is directly used within the signature, as for the case of, for instance, ULI_NMR and ULI_TAG.

The state-diagram for the federate, shown in Figure 2, is similar to the one described in the HLA standard, where the federate alternates between the "Time Advancing" and "Time Granted" states, and the triggers for transition are the TAR call and TAG callback occurring over ULI. Also, a generic ULI_Call or ULI_Callback may occur, which simply leads the federate to persist in the state it was residing in (the notation "H" is used in the HLA standard just to indicate persistence in the origin state upon the occurrence of some event). However, when TiMaC is involved, the federate execution can be suspended and resumed (in order to support Wait-Until-Commit management). It is important to note that the events that lead to suspension and resuming are not HLA callbacks (denoted by the "dagger" symbol in the HLA standard). With TiMaC, the federate is not aware of those events happening, hence its execution is oblivious that any such event has ever occurred. We extend the original HLA standard notation and denote this kind of events by using the double "dagger" symbol.

In addition to the "Suspend Federate" and "Resume Federate" events, another similar event is "Restore Federate." This event always brings the federate to the Time Advancing state. This is because the federate state is saved by TiMaC upon invocation of the NMR service over ULI, namely upon the starting of any new epoch. Hence, any rollback transparently handled by TiMaC, restores the federate to the starting point of a previous epoch.

The state-diagram for TiMaC, shown in Figure 3, alternates between the "Time Advancing" and "Time Granted" states. On the other hand, most of the computation by TiMaC is event driven, reacting to events from the RTI (over LLI) or the federate (over ULI), regardless of the state TiMaC resides in. We represent by sharp-corner rectangles computational tasks triggered by an event occurrence, which however does not trigger

a state change. Conversely, the states TiMaC can reside in are still represented by the rounded-corner rectangles. For instance, an ULI_NMR issued by the federate triggers the saving of the federate state, however TiMaC still persists in the same state it was residing in. Slightly more complex are the activities associated with a generic ULI_Call from the federate, different from ULI_NMR.

—If the call is idempotent, it is immediately issued to the RTI, according to the Immediate-Forward policy;
—If the call is nonidempotent and nonobservable, it is stored within the current-epoch log so that it can be sent to the RTI at a later time, according to the Delay-Until-Commit policy;
—If the call is nonidempotent and observable, it is stored and, according to the Wait-Until-Commit policy, the federate is suspended.

The callbacks received by TiMaC from the RTI (LLI_Callback) are treated differently depending on whether they are associated with a timestamp value or not, and on the relation between the timestamp (if any) and the time-interval characterizing the current epoch for the federate. Both types of callbacks are logged into appropriate queues (RO-log and TSO-log) upon their reception, with timestamped callbacks sorted in TSO-log according to increasing timestamp values. The callbacks whose timestamp is lower than, or equal to, the current federate granted time will trigger a rollback of the federate, while the callbacks with a higher timestamp will be delivered to the federate (via ULI_Callback) at the proper time (namely during the proper epoch). On the other hand, any LLI_Callback not associated with a timestamp value (e.g., those related to the delivery of RO messages) are simply issued towards the federate. Upon their delivery, they are also moved from RO-log to the current-epoch log, which allows restoring the RO-log in case some epoch during which these callbacks were issued towards the federate gets eventually rolled back. The management of the RR callback over ULI is somehow similar to the management of generic timestamped callbacks. In particular, if the RR callback annihilates some LLI_Callback with time less than the current-epoch upper limit, then a rollback operation is triggered. During the rollback phase, TiMaC exploits LLI_Retract in order to inform the RTI that all the previously issued idempotent calls, whose retractors are found within the logs associated with rolled-back epochs, need to be undone.

Upon the occurrence of the LLI_TAG callback, TiMaC flushes to the RTI all the calls associated with committed epochs, which require Delay-Until-Commit or Wait-Until-Commit management. It also performs memory recovery actions, in relation to the logs associated with committed epochs. Finally, it issues the "Federate Resume" event in case the federate was suspended on a call that gets committed.

As a last note, in order not to violate HLA rules, the simulation time associated with the LLI_FQR call by TiMaC needs to be selected as the minimum among the timestamp associated with the current ULI_NMR call from the federate and the timestamp of both incoming and outgoing calls and callbacks associated with uncommitted TSO messages (namely, messages delivered to, or received from, the federate during some not yet committed epoch).

The reader can refer to the Online Supplement for an alternative presentation of the TiMaC state machine in the form of pseudo-code, and for examples of execution.

6. EXPERIMENTAL STUDY

The implementation of TiMaC used in this experimental study has been developed using the `C` programming language and has been tailored for integration with the open source B-RTI package [Fujimoto et al. 2000]. Also, we used the POSIX compliant version of MASM presented and evaluated by Santoro and Quaglia [2005b]. Concerning

Table III. Computing Platform

Machine Type	RAM	OS
SMP 4 CPUs - Xeon 2.0 GHz	4 GB	Linux 2.6.8
Dual-Core - Xeon 2.8 GHz	2 GB	Linux 2.6.5
Dual-Core - Xeon 2.8 GHz	2 GB	Linux 2.6.22
Pentium-4 3.0 GHz	1 GB	Linux 2.6.8
Pentium-4 2.4 GHz	1 GB	Linux 2.6.18

functions offered by standard programming libraries, the developed implementation performs interception and Wait-Until-Commit management for I/O calls issued by the federate via `printf()` and `scanf()`.

We initially base the experimental study (see Part A) on synthetic benchmarks derived by exploiting a trace taken from a simulated demonstration exercise relying on the Joint Semi-Automated Forces (JSAF) simulation program, which has already been exploited in a number of other studies within the HLA context [Santoro and Fujimoto 2008; Tacic 2005]. Our experimental study is then concluded (see Part B) by reporting results related to runtime dynamics observed for the case of a self-federation of multiple instances of a PCS simulator. Prior entering these parts, we provide information on the used computing platform, and on the selected federate deploy policies.

6.1. Computing Platform and Federates Deploy

The experiments have been carried out on a departmental cluster formed by five heterogeneous 32-bit machines interconnected via a 100 Mb switched Ethernet LAN. Table III reports technical details for all the machines, including information on the version of the running Operating System. The machines are listed from top to bottom in the table according to the overall computing power they offer (with the top-power machine listed at the top of the table). The total amount of CPUs/cores available within the cluster is 10, while the total amount of RAM memory is 10 GB. In our experiments, each federate is hosted by a distinct CPU/core, and we take measurements for a performance comparison between conservative and TiMaC-based optimistic synchronization with an amount of federates N ranging from 2 to 10.

Given that the communication latency between federates may impact synchronization dynamics[2], we have decided to parameterize the performance study by adopting the two differentiated deploys of the federates across the cluster as described here.

1) Tight-Coupling (TC). The N federates involved in the federation are allocated on the smallest amount of machines able to host them, starting from the top-power machine and then going down towards the bottom-power machine of the cluster whenever additional CPUs/cores are required for the deploy. This policy has the advantage of reducing the communication overhead by deploying the federates as tightly coupled as possible by exploiting multiple CPUs and multiple cores of the top-power machines within the cluster.
2) Round-Robin (RR). each of the N federates involved in the federation is allocated on the less loaded machine. For example, the 2-federate case allocates one federate on the SMP 4 CPUs - Xeon 2.0 GHz machine and the other on the Dual-Core - Xeon 2.8 GHz machine. This policy has the advantage of reducing the memory load.

[2]For conservative synchronization, the communication latency affects the timeliness of the preventive detection of event safety, which is based on a (distributed) reduction to be performed by the RTI, whose completion time depends on the timely delivery of messages [McLean and Fujimoto 2003]. On the other hand, for optimistic synchronization the communication latency may affect the rollback pattern, as well as the amount of memory used [Carothers et al. 1994], which may impact the speed of simulation time commitment.

While not representative of a dedicated computing cluster with hundreds of cores, the heterogeneous computing platform used in the experiments is anyway representative of small-sized computing centers, with limited resources, whose scheduling can be still delegated to batch systems like LSF [Platform 2010] or PBS [Cluster Resources 2010]. As for this aspect, these differentiated policies allow capturing deploy instances provided by LSF/PBS like schedulers, whose decisions depend on the number of available CPUs/cores, independently of their coupling level.

6.2. Part A

6.2.1. The JSAF-Based Benchmark Application. The specific model taken by JSAF uses ad-hoc wireless communication technology to improve the effectiveness of mobility forces. Traced movement of vehicles was collected and used to drive the experiments by constructing a suite of benchmarks via the variation of the following parameters.

(NMO). The Number of Mobile Objects, selected from the trace, which are simulated within the federation. Variation of this parameter allows us to simulate regions with different mobile device populations, which gives rise to different interaction patterns among the federates in simulation time for notifying each other position updates of the hosted mobile objects via TSO messages. The value of NMO has been varied between 20 and 100.[3] This has been done by mapping 10 mobile objects to a single federate, and, as said, varying the number of federates N from 2 to 10.

(NLE). The average Number of Local Events at each federate before the notification of any new position for the mobiles occurs via TSO messages. Since local events are fictitious (in fact the trace only describes movement of mobile devices, without defining the exact context in which the movement occurs), for each local event the CPU cost has been defined according to a uniform distribution with mean value 500 microseconds. NLE has been varied between 10 and 1000, so to simulate scenarios with less or more events per simulation time unit. Also, the timestamps of the local events have been presampled so to generate final traces for the experiments by enriching the original JSAF trace, which only contains mobile update events. The pre-sampling scheme is based on a uniform distribution of the selected amount of local events over the whole simulation time span covered by the JSAF trace. The scheduling of local (fictitious) and mobility (JSAF native) events is done in a realistic fashion by explicitly maintaining and manipulating at each federate a timestamp ordered event queue containing the already scheduled events destined to the 10 mobile objects modeled by the federate. We have selected 50 as the average number of already scheduled local (fictitious) events populating the event queue and targeting each mobile object. Each time a local event destined to a mobile object is executed by the federate (and hence is extracted from the event queue), a new local event is added to the event queue, which is destined to that same mobile object. Also, the insertion of events within the event queue has been done by randomly picking pre-sampled local events currently not recorded within the event queue, with the constraint that the next local event for each mobile object is always guaranteed to be already recorded within the event queue. This has been done to maintain compliance with event causality rules proper of discrete event simulation systems. The same random approach has been used to select the 50 local events to be inserted within the event queue for a specific mobile object at simulation start-up. We note that increasing NLE, once fixed NMO, mimics the case of applications with higher CPU requirements for local processing activities, compared to resource requirements for explicit communication among the federates via TSO messages. According to a

[3]By the JSAF trace, the value 100 for NMO is the limit beyond which the need for replicating a same mobile object behavior via the trace samples might give rise to unrealistic scenarios.

different perspective, an increased value of NLE is representative of federates more loosely coupled due to reduced frequency of interactions via TSO messages per wall-clock-time unit. From the modeling point of view, NLE set to 10 gives rise to something very close to a mobility model (as expressed by JSAF-native mobility events), while for larger values of NLE the application layer mimics a simulation model entailing generic aspects, among which we also have mobility.

(NVMP). The Number of Virtual Memory Pages touched in write mode by the execution of each local, fictitious event. This parameter allows us to model application software exhibiting different memory locality patterns. Specifically, fictitious events updating a larger amount of virtual memory pages are representative of federate-level software exhibiting lower locality, which, in its turn, impacts the cost of transparent, page-based, incremental checkpointing (and recovery) performed via MASM facilities exploited by TiMaC. This allows us to stress the effects of transparent optimistic synchronization while varying the relative overhead of checkpointing vs the event execution cost. For NVMP we have selected the three different values 1, 20, and 256.

Actually, JSAF models typically require a maximum of 256 MB of RAM [Sarjoughian et al. 2001]. Considering this value as representative and assuming similar memory requirements for all the objects in our benchmarks, we obtain that the maximum memory requirement for each individual object would range from 12.8 MB, for the case of NMO set to 20, down to 2.56 MB, for the case where the model encompasses 100 mobile objects. Hence, the span of NVMP up to 256 pages (each of 4 KB) roughly means that the updated portion of the state of an object at each event, which needs to be incrementally checkpointed, spans in our study up to slightly less than 10% for the case of NMO set to 10 (which, in terms of memory requirements, mimics the case of a lower amount of larger objects within the model), and up to almost 40% for the case of NMO set to 100 (which mimics the case of a larger amount of smaller objects within the model).

(IOI). The I/O Interval, defined as the number of events between two subsequent calls to the `printf()` function issued by the application in order to log some information on a file where the standard output channel has been redirected. The introduction of this parameter is motivated by the specific policies adopted by TiMaC while supporting the mapping of conservative onto optimistic Time Management. In particular, the need for employing the Wait-Until-Commit policy for the treatment of `printf()` induces a kind of throttled execution over LLI. Hence, variation of the period for invocations to `printf()`, which has execution semantic equivalent to a nonidempotent and observable HLA service, allows us to assess the effectiveness of TiMaC in scenarios imposing more or less controlled optimism over LLI depending on the specific nature and frequency of the services invoked at the application level. In our experiments, each `printf()` outputs a string with the current simulation time value seen by the federate. Also, three different values have been used for IOI in the experiments, namely 1, 100 and INFINITE. When IOI is set to INFINITE, no invocation at all is ever issued by the federates to `printf()`, which captures the case of no throttling at all over LLI.

Finally, we note that B-RTI does not provide Data Distribution Management (DDM) services, which are explicitly oriented to controlling the volume of communication among the federates and to enhance scalability of data exchange [Raczy et al. 2005; Tan et al. 2000; Pan et al. 2007; Gu et al. 2008]. To cope with this lack, we have emulated a DDM-like scenario at the application level in order to provide results more representative of runtime dynamics that would have been obtained via, for instance, a commercial RTI implementation providing the complete suite of HLA services. In particular, each time a new position is defined for a mobile object according to the JSAF

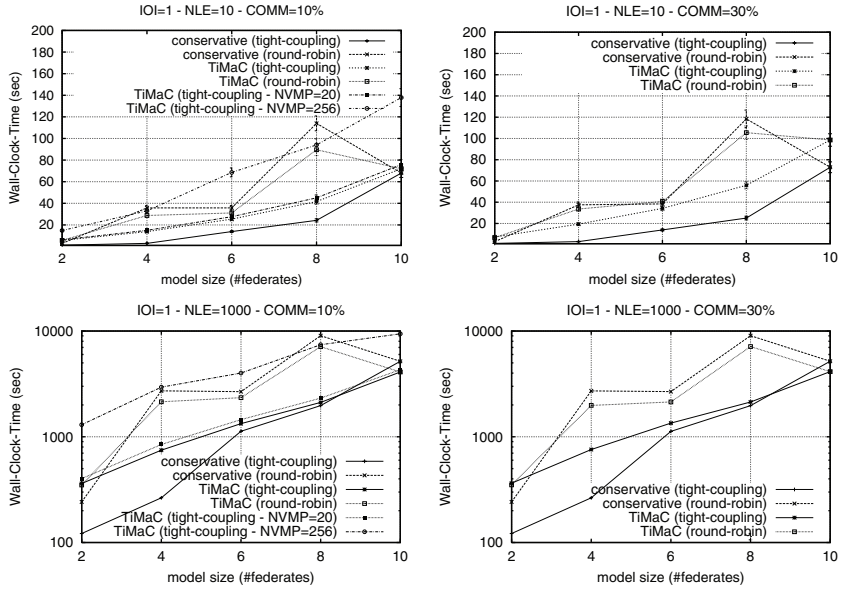

Fig. 4. Results for IOI set to 1.

trace, the corresponding notifications via UAV are not issued towards all the other mobile object (as it would occur without DDM in the case where each mobile object subscribes to each other object). Instead, they are issued towards a subset of close mobile objects, representing a percentage COMM of the whole set of mobile objects involved in the simulation. COMM has been varied from 10% to 30% in order to mimic differentiated scenarios employing differently parameterized DDM services (e.g., different interest-grid sizes [Ayani et al. 2000; Boukerche et al. 2006]). The identification of the close mobile objects allowing COMM coverage while sending position-update information, has been done off-line based on a-priori knowledge provided by the JSAF trace.

6.2.2. Results. We focus on evaluating the wall-clock-time for the execution of the benchmarks. Each reported value results as the average over 10 runs characterized by different pseudorandom seeds for the generation of the timestamps of local (i.e., non-JSAF-native) events, and of their actual CPU requirements. Also, for the cases where the y-axis is not logarithmic, we explicitly report the error bars related to the 10 samples.[4]

In Figure 4 we report the results for the case of IOI set to the value 1, while varying other parameters such as the number of federates (and hence NMO), NLE, COMM and NVMP. In this scenario we have a highly throttled execution over LLI, since the federate is allowed to enter the next epoch only after the current epoch is finalized (with ether commit or rollback). Therefore, the federate has the opportunity to optimistically process a single epoch at a time. Experiments with this settings are relevant since, although the out-coming statistics are not expected to provide indications about the full potential of TiMaC, they provide indications on the overhead by TiMaC with respect to the case of direct interactions between federate-level software and B-RTI via conservative Time Management services. Such an overhead can be correctly evaluated when the benefits by TiMaC (in terms of speculation) are eliminated via the highly throttled

[4]For logarithmic scale, the error bars are not reported for readability. However, for all the runs, the values of each individual sample was in the $\pm 7\%$ of the experimental mean value.

approach over LLI. The results however are, in some instances, surprising. For the case of federates deployed according to TC, traditional conservative synchronization exhibits performance better than TiMaC based synchronization. Indeed, the plots show a gain by conservative synchronization (in terms of reduction of the wall-clock-time for the run), which is an expected behavior since TiMaC is basically behaving similarly to conservative synchronization, but with the addition of the overhead for the management of state log/restore operations and retraction of messages, where state restore and message retraction materialize only in case the current epoch gets finalized with a rollback operation. Also, the gain by traditional conservative synchronization is more evident when NLE is set to 1000, compared to the case of NLE set to 10, and for smaller values of the number of federates. This is because, for larger values of NLE, there is a predominance of epochs entailing local processing activities only, hence the relative overhead for log/restore operations is actually higher. In other words, the weight of communication associated with the notification of new mobile positions according to JSAF-native events is relatively low. Such a phenomenon is more evident for reduced values of the amount of involved federates since the communication between the underlying B-RTI instances, natively supporting reduction operations for advancing the GALT, exhibits very reduced overhead due to the tight coupling of the CPUs/cores hosting the application according to the TC deploy. For this experiment, we also report the wall-clock-time curve for TiMaC when NVMP is set to 20 and to 256. Given that an increase of NVMP allows determining how the overhead due to state log/restore operations scales vs the size of the federate-state portions that are dirtied by event processing, we report such curves for the case of COMM=10%, where the communication overhead for mobile-position updates across the federates is reduced compared to the case COMM=30% (thus reducing the likelihood of masking variations of the state log/restore overhead vs variations of NVMP). By the plots we see that the performance curve by TiMaC for NVMP set to 20 is only slightly worse than that with NVMP set to 1 (simply marked as TiMaC), thus indicating no relevant impact of the state log/restore architecture with respect to intermediate scale-up of size of the federate-state portions dirtied by event processing. Also, although with NVMP set to the significantly scaled-up value of 256 (where 1 MB of memory is logged at each event) we observe a clear reduction of the performance of TiMaC, we anyway note that such a reduction tends to become less evident when increasing the size of the federation. Overall, the curves show a trend where the relative overhead due to transparent checkpointing scales down while increasing the size of the computing platform hosting the federation.

On the other hand, for the case of federates deployed according to RR, the performance curves of conservative and TiMaC based synchronization, although being very similar, show a reverse behavior, where TiMaC exhibits a reduction of the wall-clock-time for the run. This was not properly expected, and the reason why it happens is in that the increased amount of network (hence non-shared-memory) communication introduced by the RR deploy tends to slow down conservative synchronization more than what happens for TiMaC. In particular, advancement of the GALT, to be performed via reductions across the network, affects conservative synchronization more than TiMaC since (i) with this settings, log/restore tasks by TiMaC are likely executed fully overlapped with the distributed reduction and (ii) the single epoch speculation proper of IOI set to 1 further favors TiMaC performance. These phenomena are more relevant for NLE set to 1000, where the communication overhead for mobile-position updates results relatively less relevant, compared to the case of NLE set to 10.

Before analyzing other configurations, let us observe that RR shows a peak of wall-clock-time when the number of involved federates is set to 8, instead of having the peak at 10 federates. This does not occur for TC and is due to the combination of the following two effects: (i) Compared to TC, RR leads to an increase of the message delivery delay,

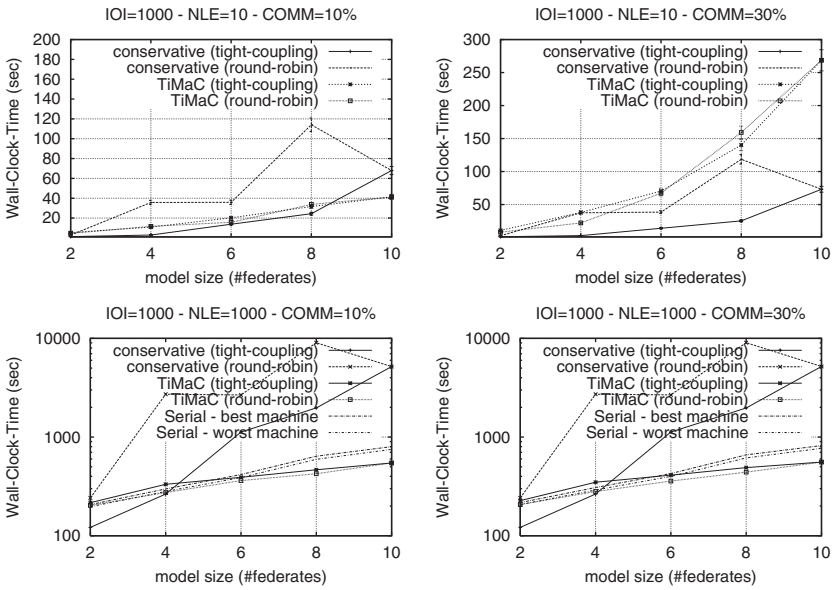

Fig. 5. Results for IOI set to 1000.

which may in turn delay local processing activities. Particularly, the amount of remote communication for 8 federates deployed according to RR is almost the same as the one for 10 federates, still deployed according to RR. (ii) RR leads to suboptimal mapping over the computing platform. Specifically, for the 8-federate case, only the 25% of the simulation load runs on the top-power machine of the cluster, while for the 10-federate case, up to 40% of the simulation load is running on that machine.

Figure 5 shows the results for IOI set to 1000. For this setting, throttling over LLI when TiMaC is employed is definitely reduced, since up to 1000 epochs are allowed to be speculatively processed before the call to `printf()` triggers the Wait-Until-Commit policy. Hence, these curves are representative of contexts where (differently from the case of IOI set to 1) TiMaC based speculation is actually exploited, thus leading to runs entailing a significant amount of optimism. A first observation by the plots is related to the fact that, once fixed the run settings, TiMaC exhibits very similar performance regardless of the employed deploy scheme (RR or TC). This is exactly aligned with the nature of optimism oriented synchronization (see, e.g., the results in [Carothers et al. 1994]), which tends to exhibit runtime dynamics moderately sensible to differences in the communication delay caused by different deploy choices or other factors. This is not the case for conservative Time Management that, similarly to the case of IOI set to 1, exhibits performance definitely dependent on the selected deploy policy. We also note that, when NLE is set to 1000, TiMaC shows performance comparable to the one by conservative synchronization when the deploy is TC and the number of federates is smaller than or equal to 4. On the other hand, for the RR deploy and/or when the number of federates is greater than 4, TiMaC significantly reduces the wall-clock-time for the run compared to conservative Time Management. Independently of the value of COMM, the maximum wall-clock-time reduction by TiMaC is on the order of 90% for TC (see the 10-federate case) and on the order of 95% for RR (see the 8-federate case). For NLE set to 1000, we also report the performance curve achieved by executing the simulation model according to a sequential-like fashion (with all the mobile objects involved in the simulation hosted by the same instance of federate, which runs over

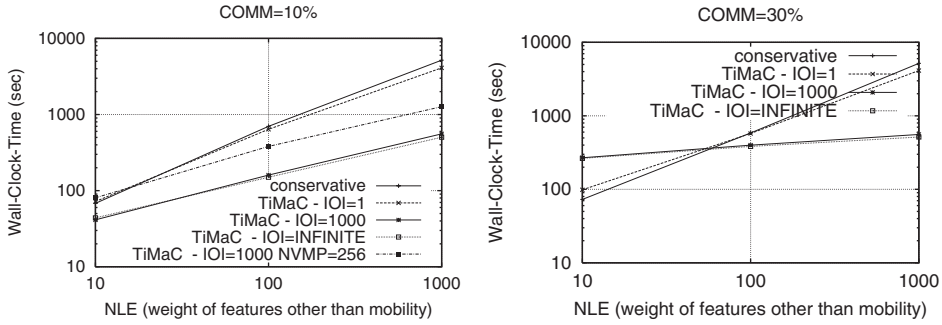

Fig. 6. Results vs NLE, IOI and NVMP for NMO=100 (10 federates).

a single instance of B-RTI as a stand alone component). For the sequential-like run, we report two different performance curves, which we refer to as best/worst machine, since they express the best/worst performance data observed for the sequential-like run on the various machines of the cluster. By these data we observe that, while increasing the model size/complexity, the performance of conservative Time Management rapidly degrades, thus not even justifying the partitioning of the simulation model into distinct federates. In fact, the sequential-like run definitely outperforms conservative Time Management as soon as the number of mobile objects is greater than 40 (resp. 20) when the RR (resp. TC) deploy is adopted. On the other hand, TiMaC based synchronization better copes with scaled-up model size, while still not penalizing performance for reduced model size. In particular, when the number of simulated mobile objects is greater than 60, TiMaC based model execution with the mobile objects partitioned across different federates allows a reduction of the wall-clock-time for the run, up to a maximum of 33% for 100 mobile objects partitioned across 10 federates. Overall, TiMaC shows better scalability compared to the sequential-like approach. The only exception to these considerations can be noted when NLE is set to 10 and COMM is set to 30%. For this setting, the performance of TiMaC based synchronization rapidly degrades when the number of federates is greater than or equal to 8. The reason is in that, for NLE=10 and COMM=30%, the run mimics a model execution with high communication requirements across the federates, compared to local processing activities. Hence, TiMaC is prone to rollback thrashing effects due to the overhead of both log/restore and, more important, message retractions. However, in order to cope with "pathological" cases entailing a significantly reduced computation-to-communication ratio, the TiMaC state machine could be complemented by solutions like, for instance, the one by Quaglia and Santoro [2007], which allows reducing the impact of message retractions via the artificial delay of message send operations towards the RTI, in the hope that a retraction, if required, is intercepted and processed locally.

Figure 6 provides a comparison between conservative Time Management and TiMaC while focusing on the largest model size. In particular, we report wall-clock-time values once set NMO to 100, with the mobile objects distributed across 10 federates, and while varying both NLE, IOI, NVMP and COMM. Given that we are running 10 federates on the whole set of the 10 CPUs/cores available within the cluster, the actual deploy is independent of the adopted deploy policy (TC or RR). For COMM set to 10%, we observe that the performance curve by TiMaC is almost identical to the one by conservative synchronization for IOI set to 1. We recall that value 1 for IOI means minimal speculation, but again provides a scenario where the overhead by TiMaC, compared to conservative synchronization, can be assessed, which appears again very limited (or null). On the other hand, as soon as we have an increase of the value of IOI,

TiMaC definitely outperforms conservative synchronization. In particular, for IOI set to 1000 or INFINITE, TiMaC provides wall-clock-time reduction ranging from about 50% (when NLE is set to 10) up to 90% (when NLE is set to 1000). This indicates how the performance provided by TiMaC better scales vs the model size/complexity expressed in terms of, for instance, number of involved federates. Such a trend is still verified even when considering a model with largely scaled up requirements in terms of memory pages to be logged at each event, namely when NVMP is set to 256 (instead of 1). Similar considerations can be made for the case of COMM set to 30%, with the only exception that the performance gain by TiMaC starts to be noted for values of NLE close to 100. In fact, when COMM=30%, very low values of NLE give rise to reduced computation-to-communication ratio which is, in general, adverse to speculation based schemes, and could be effectively supported by complementing TiMaC with approaches aiming at artificially inducing throttling.

6.3. Part B

In this second part of the experimental study we consider the case of a self-federation formed by multiple instances of a same simulator. We note that the self-federated approach is a interesting case study since it mimics, within the HLA context, the kind of parallelization that is typical of PDES (Parallel Discrete Event Simulation) approaches. In particular, self-federating a simulator has resemblances with PDES contexts where multiple homogeneous Logical Processes are involved within the simulation run. Hence, this scenario allows determining runtime dynamics related to the scaling up of the size of a specific simulation model.

The experiments have been carried out by self-federating a PCS simulator modeling fixed base stations offering communication services to mobile devices and performing power regulation based on the Signal-to-Interference Ratio (SIR) evaluated considering cross channel interference within a same cell [Kandukuri and Boyd 2002]. Each federate is in charge of simulating a coverage area formed by a group of 256 microcells, each one managing 100 wireless channels. The movement of mobile devices follows a classical random walk model [Akyildiz et al. 2000] with cell switch time exponentially distributed with mean 5 minutes, and the average call holding time has been set to 2 minutes. Each federate includes the modules for pseudo-random generation of call arrivals within the corresponding group of cells so that the interaction between different federates takes place only in case of hand-off occurrence for a mobile involved in an ongoing call, which switches between cells in different groups. The threshold SIR to be achieved by power regulation at the start of a call has been set at about 10 DB, similarly to what happens in standard GSM transmission, and the inter-arrival time of calls per each cell follows an exponential distribution [Boukerche et al. 1999; Carothers et al. 1994, 1995] with mean value selected to achieve channel utilization factor of about 60%.

Figure 7 (left side) shows the execution speed of the simulation, evaluated in terms of simulated-time units per wall-clock-time unit, for the two different investigated configurations (i.e., conservative synchronization and optimistic synchronization via TiMaC), while varying the number of federates between 2 and 10. For this set of experiments, we have adopted the TC federate deploy. Like for the study proposed in Part A, each reported value is the average of ten samples. Further, although not explicitly reported within the plots, the measured values were dispersed in the interval $\pm 5\%$ around the mean value.

In order to provide further hints on the runtime of TiMaC vs conservative synchronization, we also report the execution speed for the case where the simulation model explicitly makes use of lookahead. In particular, each of the aforementioned ten runs has been pre-sampled to determine the minimum simulation time interval between

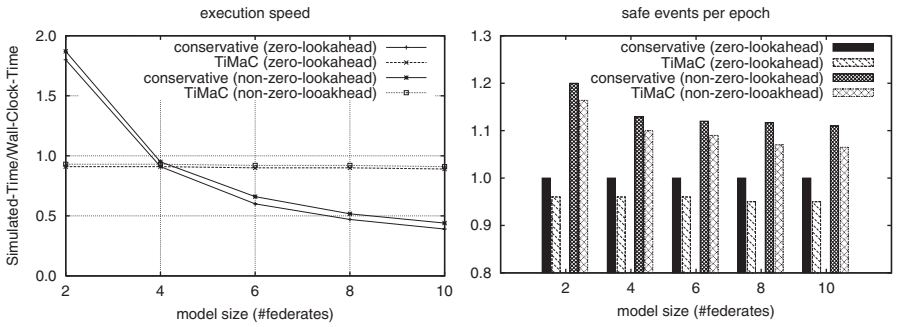

Fig. 7. Results for self-federation of PCS simulators.

events addressed to remote federates, which has been then used as the lookahead value for the corresponding run giving rise to the actual sample considered in the statistics.

From the data we observe how the conservative approach provides performance values that strongly vary vs the size of the federation. In particular, it allows for 97% to 100% enhancement of the execution speed compared to TiMaC for small federations (i.e., for the two-federate case), depending on whether lookahead is exploited or not. However, as soon as the size of the federation gets increased, which implies making use of less tightly coupled resources (even under the TC deploy), the performance of conservative synchronization degrades, thus leading TiMaC to exhibit a performance gain up to 225% for the case of zero-lookahead and 200% for the case of reliance on nonzero lookahead (see the configurations with 10 federates).

We finally report in Figure 7 (right side) some data related to the number of safe (i.e., non-rolled back) events for epoch. This number corresponds to the amount of events safely processed by each federate in the interval between two successive TAG callbacks, issued either by the RTI (in case of conservative synchronization) or by TiMaC (in case of transparent optimistic synchronization). The number of safe events per epoch is clearly equal to 1 for the case of conservative synchronization when zero-lookahead is adopted. Indeed, no event is ever rolled back by this synchronization scheme, and each federate always asks for advancement to the logical time associated with the next to be processed event in the local event queue, hence giving rise to runtime dynamics where a single event is actually processed for each TAG. On the other hand, for the case of nonzero lookahead the number of safe events per epoch for conservative synchronization increases, by a maximum of 20%, as observed for the two-federate case. By the reported values we see how, independently of whether the lookahead is zero or not, TiMaC based runs guarantee an amount of safe events per epoch which is on the order of 94-96% of the corresponding values achieved by conservative synchronization. Overall, TiMaC allows for fruitful exploitation of parallelism, since speculation highly likely gives rise to correct computation, which is reflected in actual performance improvements while increasing the federation size, as discussed previously.

7. CONCLUSIONS

In this article we have have provided a mapping of conservative onto optimistic HLA synchronization services via a state machine referred to as Time Management Converter (TiMaC), which can be transparently interposed between federate-level code and the underlying RTI middleware component. Our proposal is in the direction of allowing simplified programming for the federate code, by letting it rely on the very intuitive conservative Time Management services, while transparently enabling optimistically

synchronized runs. TiMaC has been designed on the basis of a classification and treatment policies we have defined for HLA-services. An experimental study of the effects of TiMaC has been finally provided, which has been based on a suite of benchmarks derived by exploiting traces available from the Joint Semi-Automated Forces simulation program, and on a self-federation of Personal Communication System (PCS) simulators. Interesting findings by the experimental results include better scalability of TiMaC, compared to conservative Time Management, vs the size of the federation and reduced performance sensibility vs the level of coupling among the involved federates depending on the deploy over the computing platform.

ELECTRONIC APPENDIX

The electronic appendix to this article is available in the ACM Digital Library.

REFERENCES

Akyildiz, I. F., Lin, Y. B., Lai, W. R., and Chen, R. J. 2000. A new random walk model for PCS networks. *IEEE J Select. Areas Comm. 18,* 7, 1254–1260.

Ayani, R., Moradi, F., and Tan, G. S. H. 2000. Optimizing cell-size in grid-based DDM. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*. IEEE, 93–100.

Bagrodia, R., Chandy, K. M., and Liao, W. T. 1991. A unifying framework for distributed simulation. *ACM Trans. Model. Comput. Simul. 1*, 348–385.

Boukerche, A., Das, S. K., Fabbri, A., and Yildz, O. 1999. Exploiting model independence for parallel PCS network simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*. IEEE, 166–173.

Boukerche, A., Dzermajko, C., and Lu, K. 2006. An enhancement towards dynamic grid-based DDM protocol for distributed simulation using multiple levels of data filtering. *Parallel Comput 32,* 11–12, 902–919.

Cai, W., Turner, S. J., Lee, B.-S., and Zhou, J. 2005. An alternative time management mechanism for distributed simulations. *ACM Trans. Model. Comput. Simul. 15,* 2, 109–137.

Carothers, C. D., Fujimoto, R. M., and England, P. 1994. Effect of communication overheads on Time Warp performance: an experimental study. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*. IEEE, 118–125.

Carothers, C. D., Fujimoto, R. M., England, P., and Lin, Y. B. 1994. Distributed simulation of large-scale PCS networks. In *Proceedings of the 2nd IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, 2–6.

Carothers, C. D., Fujimoto, R. M., and Lin, Y. B. 1995. A case study in simulating PCS networks using Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*. IEEE, 87–94.

Chen, D., Turner, S. J., Cai, W., Gan, B.-P., and Low, M. Y.-H. 2005. Algorithms for HLA-based distributed simulation cloning. *ACM Trans. Model. Comput. Simul. 15,* 4, 316–345.

Cluster Resources, I. 2010. Torque resource manager. http://www.clusterresources.com/products/torqueresource-manager.php.

Das, S. R. and Fujimoto, R. M. 1997. An empirical evaluation of performance-memory trade-offs in Time Warp. *IEEE Trans. Parallel Distrib. Syst. 8,* 2, 210–224.

DMSO. 2004. Runtime Infrastructure (RTI). https://www.dmso.mil/public/transition/hla/rti.

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

Fujimoto, R. and Weatherly, R. M. 1996. Time management in the DoD High Level Architecture. In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*. IEEE, 60–67.

Fujimoto, R. M., McLean, T., Perumalla, K. S., and Tacic, I. 2000. Design of high performance RTI software. In *Proceedings of the 4th International Workshop on Distributed Simulation and Real-Time Applications*. IEEE, 89–96.

Georgia Tech Research Corporation. 2003. FDK - Federated Simulations Development Kit. http://www.cc.gatech.edu/computing/pads/fdk/.

Gu, Y., Boukerche, A., and de Araujo, R. B. 2008. Performance analysis of an adaptive dynamic grid-based approach to data distribution management. *J. Parallel Distrib. Comput. 68,* 4, 536–547.

IEEE. 2000a. IEEE STD 1516-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules.

IEEE. 2000b. IEEE STD 1516.1-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface (FI) Specification.

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

Jha, V. and Bagrodia, R. L. 1994. A unified framework for conservative and optimistic distributed simulation. *SIGSIM Simul. Dig. 24*, 12–19.

Kandukuri, S. and Boyd, S. 2002. Optimal power control in interference-limited fading wireless channels with outage-probability specifications. *IEEE Trans. Wirel. Comm. 1,* 1, 46–55.

McLean, T. and Fujimoto, R. M. 2003. Predictable time management for real-time distributed simulation. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*. IEEE, 89–96.

Pan, K., Turner, S. J., Cai, W., and Li, Z. 2007. An efficient sort-based DDM matching algorithm for HLA applications with a large spatial environment. In *Proceedings of the 21st Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 70–82.

Pellegrini, A., Vitali, R., and Quaglia, F. 2009. Di-DyMeLoR: Logging only dirty chunks for efficient management of dynamic memory based optimistic simulation objects. In *Proceedings of the 23rd Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 45–53.

Perumalla, K. S. 2005. μ-sik: A micro-kernel for parallel/distributed simulation systems. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS '05)*. IEEE, 59–68.

Plank, J., Beck, M., and Kingsley, G. 1995. Libckpt: Transparent checkpointing under UNIX. In *Proceedings of the USENIX Winter Technical Conference*. USENIX Association, 213–223.

Platform. 2010. Platform LSF. http://www.platform.com/workload-management/high-performancecomputing/lp.

Quaglia, F. and Santoro, A. 2007. Transparent risk-free synchronization in the High-Level-Architecture interoperability standard. In *Proceedings of the 12th IEEE Symposium on Computers and Communications*. IEEE, 995–1000.

Raczy, C., Tan, G. S. H., and Yu, J. 2005. A sort-based DDM matching algorithm for HLA. *ACM Trans. Model. Comput. Simul. 15,* 1, 14–38.

Rajaei, H., Ayani, R., and Thorelli, L.-E. 1993. The local Time Warp approach to parallel simulation. *SIGSIM Simul. Dig. 23,* 1, 119–126.

Reynolds, JR., P. F. 1988. A spectrum of options for parallel simulation. In *Proceedings of the Winter Simulation Conference*. Society for Computer Simulation, 325–332.

Ronngren, R., Liljenstam, M., Ayani, R., and Montagnat, J. 1996. Transparent incremental state saving in Time Warp parallel discrete event simulation. In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*. IEEE, 70–77.

Santoro, A. and Fujimoto, R. M. 2008. Offloading data distribution management to network processors in HLA-based distributed simulations. *IEEE Trans. Parallel Distrib. Syst. 19,* 3, 289–298.

Santoro, A. and Quaglia, F. 2005a. Transparent state management for optimistic synchronization in the High Level Architecture. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 171–180.

Santoro, A. and Quaglia, F. 2005b. A version of MASM portable across different UNIX systems and different hardware architectures. In *Proceedings of the 9th International Symposium on Distributed Simulation and Real-Time Applications*. IEEE, 35–42.

Santoro, A. and Quaglia, F. 2006. Transparent optimistic synchronization in HLA via a Time-Management converter. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 193–200.

Sarjoughian, H. S., Hild, D. R., Hu, X., and Strini, R. A. 2001. Simulation-based SW/HW architectural design configurations for distributed mission training systems. *Simulation 77,* 1-2, 23–38.

Srinivasan, S. and Reynolds, Jr., P. 1998. Elastic time. *ACM Trans. Model. Comput. Simul. 8,* 2, 103–139.

Steinman, J. 1993. Incremental state saving in SPEEDES using C plus plus. In *Proceedings of the Winter Simulation Conference*. Society for Computer Simulation, 687–696.

Steinman, J. S. 1992. SPEEDES: A multiple-synchronization environment for parallel discrete event simulation. *Int. J. Comput. Simul.* 251–286.

Tacic, I. 2005. Efficient synchronized data distribution management in distributed simulations. Ph.D. thesis, Atlanta, GA. AAI3170113.

Tan, G. S. H., Zhang, Y., and Ayani, R. 2000. A hybrid approach to data distribution management. In *Proceedings of the 4th International Workshop on Distributed Simulation and Real-Time Applications*. IEEE, 55.

VARDANEGA, F. AND MAZIERO, C. 2000. A generic rollback manager for optimistic HLA simulations. In *Proceedings of the 4th International Workshop on Distributed Simulation and Real-Time Applications*. IEEE, 79–85.

VIRTUAL TECHNOLOGY CORPORATION. 2004. RTI NG ProTM version 2.0.2. http://www.virtc.com/Products/prdFulltext.jsp?ID=1z RTI.

WANG, X., TURNER, S. J., LOW, M. Y.-H., AND GAN, B.-P. 2005. Optimistic synchronization in HLA-based distributed simulation. *Simulation 81,* 4, 279–291.

WEST, D. AND PANESAR, K. 1996. Automatic incremental state saving. In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*. IEEE, 78–85.

Received February 2011; revised March 2011, October 2011, June 2012; accepted August 2012