2010 12th International Conference on Computer Modelling and Simulation

A Scalable HLA RTI System based on Multiple-FedServ Architecture

Ding-Yong Hong, Fang-Ping Pai, Shih-Hsiang Lo and Yeh-Ching Chung
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.
{dyhong,fppai,albert}@sslab.cs.nthu.edu.tw, ychung@cs.nthu.edu.tw

Abstract—**A scalable and high performance RTI (Runtime Infrastructure) system implements a two-layer architecture to supporting large-scale simulation is proposed in this article. The two-layer architecture, Multiple-FedServ, exploits both centralized and distributed way to manage a simulation. In the first layer, each FedServ is in charge of a number of federates and in the second layer, a simulation federation is then formed by all the FedServs. This paper describes how the messages are routed and synchronization performed in this two-layer architecture. An RTI system based on Multiple-FedServ architecture and follows the specification of IEEE 1516 standard is implemented. Performance evaluations of this RTI using standard HLA/RTI benchmarks are presented. We evaluate the latency, throughput and time advancement benchmarks under varied size of federates and varied size of FedServs. Issues such as message routing, multicasting and synchronization are specially addressed in this article. Results show that Multiple-FedServ architecture can scale well.**

Keywords- High Level Architecture, Runtime Infrastructure, Multiple-FedServ, Scalability, Large-scale Simulation.

I. INTRODUCTION

In the 1990s, Defense Modeling and Simulation Office (DMSO) [10] first proposed High Level Architecture (HLA) in order to promote reusability and interoperability of military simulations. HLA has been approved as IEEE 1516 standard in September 2000. There are three necessary elements defined in HLA, the rules, interface specification and object model template (OMT) [7]. According to the HLA rules users can develop cooperating simulations, called federates. Federates can interact with each other by invoking sets of management service provided by Runtime Infrastructure (RTI). The integrated virtual environment composed of federates is called federation. Recently, besides military applications, many simulations such as traffic simulations, factory workflow simulations and so forth use the HLA as simulation architecture.

The RTI system provides a set of management services defined in the HLA interface specification. The execution management, time synchronization, data and event exchange of the federation are all handled by the coordination of these services. As the scale of simulation grows, the scalability issue of RTI is becoming more and more important especially for wide-ranging and large-scale complex simulations. This article will focus on the scalability problems of RTI and will address the architecture we adopted to implement our scalable RTI.

The related works reported in the literature rarely take account of the scalability issues of an RTI system. Although [3], [9] and [12] propose the schemes for simulation users to easily construct a large-scale HLA simulation in terms of federate arrangement and resource management. Among these RTI systems, a centralized management is used to conduct the RTI services. That is, the RTI system uses a centralized manager or server to carry out the RTI services, such as joining and resigning of federates, data exchange for participating federates, etc. For a federation execution with a large amount of federates, the centralized manager tends to be overloaded because of busy processing the requests of the federates. In this situation, the centralized management will degrade the simulation performance. Also, it may cause a scalability bottleneck of the whole RTI system and limits the capacity of how many federates that can participate in a federation.

Considering the loading of a RTI system, two requirements are necessary to build a scalable RTI system for a large-scale simulation. One is that the RTI system must have enough capability to manage the federation execution. The other one is that the overloaded situation must be avoided. In order to achieve these two requirements, we design a hybrid architecture called Multiple-FedServ. First, we adopt a centralized server called FedServ as the basis to provide RTI services federates belongs to this server. The responsibility of a FedServ is to manage the joining and resigning of federates, and to facilitate data exchange among participating federates, etc. Since it has the central control of the system, the performance of the centralized management is better than the distributed management. Then we make several replicas of the FedServ and these FedServs will form a Multiple-FedServ RTI system to serve all federates within the federation. The RTI system becomes more powerful to serve more federates simply by integrating the capability of each FedServ. These FedServs cooperate in forming a powerful RTI system to manage all the federates. Moreover, the loading of managing a federation execution can be distributed to these FedServs. This prevents any FedServ from being overloaded.

The proposed Multiple-FedServ architecture has several advantages. First, our architecture can provide scalability and also preserve the performance. Second, based on our connection scheme, the system is well-organized and suitable for a large-scale simulation. Third, the Multiple-FedServ design is helpful for the operations of multicast and synchronization with numerous federates.

978-0-7695-4016-0/10 $26.00 © 2010 IEEE
DOI 10.1109/UKSIM.2010.102

Based on the proposed architecture, we have implemented a RTI system following the specification of IEEE 1516 standard. Moreover, we have evaluated our RTI system using the benchmarks from DMSO. The benchmarks are modified to meet IEEE 1516 standard. We test latency, throughput and time advancement benchmark under varied size of federates and varied size of FedServs. The performance of multicast and synchronization are also assessed. The results show that Multiple-FedServ architecture can scale well.

The rest of this paper is organized as follows. In the next section, we state the related works to compare different designs of RTI system. Section 3 clearly illustrates the overall design of our Multiple-FedServ RTI system architecture. In section 4, the evaluation results are provided. Finally, section 5 concludes this paper.

II. RELATED WORK

DMSO RTI [10] is a well-known RTI system on which some research works [2][9] [11][12] are based. The DMSO RTI consists of the RTI Executive Process (RtiExec), the Federation Executive process (FedExec) and libRti library to run HLA simulations. The RtiExec is a globally process to manage the creation and destruction of federation execution. The FedExec coordinates data exchange and operations among the participating federates of the same federation. The libRTI is a library that provides RTI services to federate developers. In the DMSO RTI, a federation is only managed by a single FedExec.

PADS research group at Georgia Tech provides Federated Simulations Development Kit (FDK) [8] for RTI developers to implement their specific RTI system. FDK comprises RTIKIT and two RTI implementations, BRTI and DRTI, which are built on RTIKIT. RTIKIT consists of some well-developed modules for varied network systems or architecture types, such as TCP, Myrinet [5], SGI, Linux. In the BRTI or DRTI, each federate is not only responsible to do simulation but also to be involved in the management of a federation execution. To run a federation execution needs to specify the federate IDs and IP addresses for the participating federates in the same federation. And these federates will form a fully connected simulation system.

Recent advances in Grid technologies [4] have been used to facilitate distributed resource management, dynamic load balancing, and automated configuration for the HLA simulations [3][9][11][12]. In [3], Cai et al. proposed a load management system (LMS) over the Grid. LMS exploits the Grid services to achieve load balancing, fault tolerance and security for running large-scale HLA-based simulations. In [11], Zajac et al. presented the migration service for HLA simulations under a Grid management system. The service can monitor the execution of the federate, discover available resources and then migrate the federate to appropriate resource automatically. In [9], it addresses some research issues in how to construct a large-scale simulation on the Grid, including collaborative development model development, resource management and automated federate discovery and configuration. In [12], the HLA distributed simulation proposed is built on top of Grid services which help users conveniently compose a large-scale simulation, such as index service, RTI factory service and federate factory service. Grid technology actually complements the weak parts of the HLA.

III. MULTIPLE-FEDSERV RTI SYSTEM

In the following, we will describe our hybrid HLA simulation environment in detail.

A. Multiple-FedServ Architecture

In the Multiple-FedServ architecture as shown in Figure 1, FedServs and federates are partitioned into several groups. Each group consists of one FedServ and some federates. FedServ is responsible for providing RTI services for federates in the same group, e.g. synchronization of federates, publishing/subscription, message send/receive etc.. Besides, all FedServs also cooperatively manage the execution of the whole federation. In order to coordinate the federation, the FedServs need to synchronize information with each other when states update. For example, when a federate joins the simulation or a federate publishes an object class, the corresponding FedServ will synchronize this information with other FedServs. Through this operation, every FedServ will obtain the latest and consistent information.

To start a federation execution, all federates need to establish network connections so that they can communicate with each other. In our design, a simple but efficient way to connect all federates is used: by creating network connections between all FedServs, there will exist a network route between any two federates. That is, we establish a fully-connected network among all FedServs to connect all groups. With this connection scheme, the federates are able to communicate with each other. The reason why not to connect the federates directly is that the process to build up one-to-one connection between any two federates is cumbersome. When a federate wants to join a federation, it has to collect the IP addresses and port numbers of all other federates and then connects to them one by one. As the numbers of federates joining the simulation become large, it will take a long time to finish the joining process. Moreover, maintaining a large amount of connections in a distributed system will reduce the system scalability. Therefore, we choose to connect all FedServs instead of connecting all federates directly.

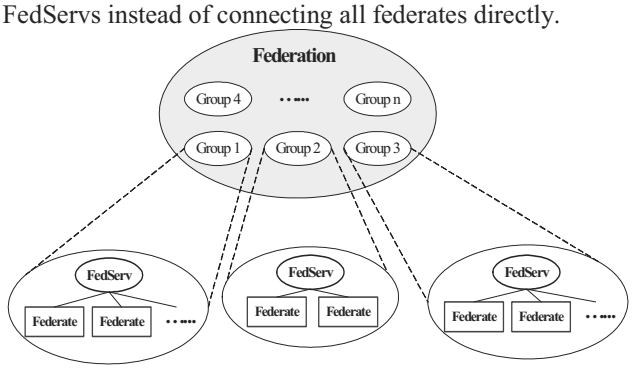

Figure 1. Multiple-FedServ Architecture.

In our implementation, the decisions of how many FedServs are in a federation and how to connect the FedServs and federates are determined by the users. The users have the best knowledge about the behavior and execution loading of the simulation programs. Therefore, the users can have the best way to arrange the execution loading. As the numbers of federates increase in the runtime, the system can still execute well by increasing the numbers of FedServs.

B. Two-layer ID Assignment Scheme

When a federate joins a federation, the RTI system must assign the federate a unique ID in the simulation environment. For a single FedServ design, the centralized FedServ must serve the ID assignment requests from all the federates. For a fully distributed design, the system must enter a global locking state to decide the next available globally unique ID. When a simulation is running with considerable federates, neither centralized nor fully distributed design is suitable for the high-scalability simulation requirement.

In our Multiple-FedServ architecture, the assignment service is provided by these FedServs cooperatively. In order to efficiently get a globally unique ID, we use a two-layer ID assignment scheme. When a federate joins a FedServ, the FedServ will assign it a pair of numbers of the form (*FedServ_ID*, *Local_Federate_ID*), where the *FedServ_ID* is the ID of the FedServ and the *Local_Federate_ID* is a unique ID in the group of the FedServ. After the assignment process, the FedServ broadcasts the ID to other FedServs so every FedServ can know that a new federate has joined the federation. The following describes how to generate the *FedServ_ID* and *Local_Federate_ID* for FedServs and federates, respectively.

The first FedServ created in a federation execution becomes the master FedServ and it will be assigned to ID 0. The other FedServs created after master FedServ in the same federation will become slave FedServs with ID starting from 1. In figure 2, there are one master FedServ and two slave FedServs which are assigned to ID 0, 1, and 2 respectively, and three federates connect to Master FedServ, two federates connect to Slave FedServ 1 and two federates connect to Slave FedServ 2.

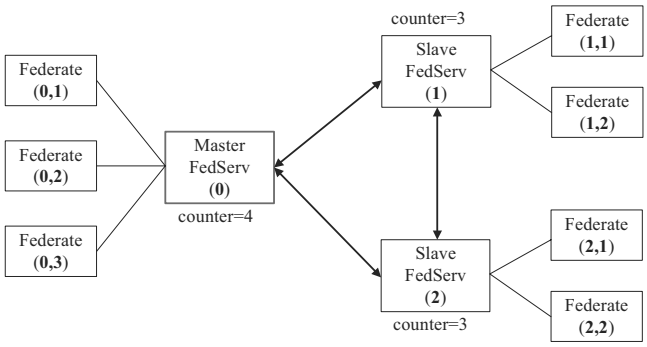

Figure 2. An example of two-layer ID assignment.

To generate *Local_Federate_ID* for each federate, every FedServ maintains a local federate ID counter starting from 1. Each time a federate joins a FedServ, it will concatenate its FedServ ID and the value of local federate ID counter as the ID of the newly joint federate and then increases the number of the counter by one. With this mechanism, we can make sure that any federate ID is certainly globally unique. For example, in figure 2, assume the three federates in the left-hand side join the federation with the order from the top to the bottom. Initially the local federate counter ID is set to 1. When the Master FedServ receives the joining request from the top-left federate, it assigns its ID to (0,1) and increases the counter to 2. This assignment process is repeated when the mid-left and bottom-left federates join it and finally the value of counter is 4. The same assignment process is executed in the groups of the FedServs in the right-hand side. We can see that no any two federate IDs will be the same.

The two-layer ID assignment scheme has the following advantages: First, the assignment process is finished locally in each group so the joining process is efficient. Second, every FedServ only has to handle the joining requests from parts of federates. This design can reduce the loading of each FedServ. Third, no any global locking is required among the FedServs. The overhead of global locking is avoided. Finally, the FedServs in different groups can handle different joining requests simultaneously. The performance is improved especially when a lot of federates are joining the federation at the same time.

C. Message Routing and Multicast

When a federate requests the RTI services of *UpdateValue* or *SendInsteraction*, the message has to be multicast to those federates which are interested in this event. The multicast operation consists of the following steps.

1. The source federate sends the message to the FedServ in the same group. This FedServ becomes the source FedServ.
2. The source FedServ determines the federates that will receive the message according to the publish/subscribe relationship.
3. The source FedServ analyzes the two-layer IDs of the destination federates and finds the destination FedServs that consist of these destination federates by the *FedServ_ID*.
4. The source FedServ sends the message to each destination FedServ.
5. Every destination FedServ determines the local federates which will receive the message according to the publish/subscribe relationship.
6. The destination FedServ sends the message to those local federates.

From the operation in step 3, the message is transferred between two FedServs only once and the network traffic is reduced. Moreover, the operation of multicast from the destination FedServs to the destination federates in different groups can be done in parallel. Therefore, the Multiple-FedServ architecture is efficient for the multicast operation.

The time to finish the multicast operation is significantly reduced especially when the destination federates are distributed to different FedServs.

To further improve the performance of the operation of multicast, we make each federate not only connect to the FedServ, but also connect to other federates within the same group. When the destination federates and the source federate are in the same group, the message is sent from the source federate to the destination federate directly instead of being forwarded by the FedServ. The loading of the FedServ is reduced. With this connection scheme, the transmission within the same group can be finished with one-hop communication

For example, in figure 3, assume the federates (0,2),(0,3),(1,2),(1,3),(2,1),(2,2) are interested in the event of the federate (0,1). When federate (0,1) updates a value, the message is first sent to FedServ 0. FedServ 0 looks up the publish/subscribe tree and determines the destination federates that are managed by other FedServs. Then, the four federates (1,2),(1,3),(2,1),(2,2) are found and divided into two groups with FedServ_ID 1 and 2. According to the FedServ_ID, FedServ 0 sends one copy of the message to FedServ 1 and one to the FedServ 2. At the same time, the federate (0,1) can send the message to federates (0,2),(0,3) directly because they are in the same group. After receiving the message, FedServ 1 and 2 look up the publish/subscribe tree and determine the federates that will receive the message in its own group. FedServ 1 and 2 send the message to federates (1,2),(1,3) and federates (2,1),(2,2) respectively and then the multicast operation is completed.

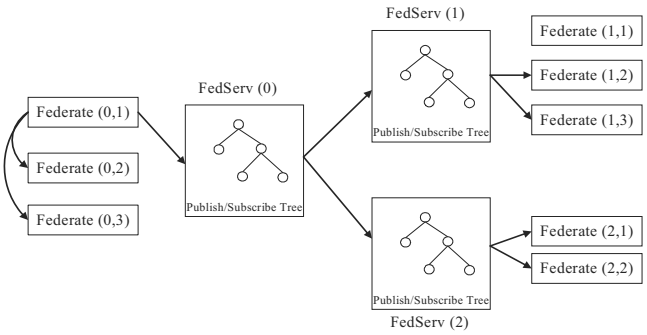

Figure 3. An example of the operation of multicast.

D. Synchronization

Synchronization is very important in the simulation environment to coordinate the execution of the simulation processes. With large amount of federates, it sometimes costs a lot of time to synchronize the federates and causes the system not to scale well. To provide an easy and efficient operation of synchronization in the Multiple-FedServ architecture, we use a two-phase synchronization scheme. The details of the two-phase synchronization scheme are described as follows:

- Phase 1: In each group, the federates start the operation of synchronization. After the federates in a group finish the synchronization, the federate with the smallest *Local_Federate_ID* will notify the FedServ that the local synchronization is done.
- Phase 2: The FedServs start the operation of synchronization after they receive the notification from their local federate. When the FedServs finish the synchronization, all federates are synchronized. Finally, each FedServ triggers the callback to the federates in its own group.

To improve the performance of the two-phase synchronization, we adopt the butterfly barrier [1] in both phases of the operation of synchronization. Because we have connected the federates in the same group and all the FedServs are also connected, the butterfly barrier can be performed in both phases. Besides, the synchronization operation of phase 1 in each group can be performed in parallel.

IV. Performance Evaluation

This section demonstrates the performance results of our implementations. The FedServs and federates evenly run on 6 IBM e-Servers. Each IBM e-Server has dual Xeon 2.4 GHz CPUs, 1 GB DRAM and runs Linux operating system with kernel version 2.6.8. All machines are connected with Gigabit Ethernet network. The RTI system is evaluated by using the benchmarks from DMSO. The benchmarks are modified to meet the IEEE HLA 1516 standard and the maximum message size allowed is increased so the performance with large size of message payload can be tested. We evaluate the latency, throughput and time advancement benchmarks under varied size of federates and varied size of FedServs. We also test the performance of the operations of multicast and synchronization.

A. Latency and Throughput Benchmark

The latency program measures RTI performance in terms of the elapsed time from sending to receiving an attribute update. The latency measurement is processed by two federates. First, one federate sends an attribute value to the other federate. The other federate is responsible to send this attribute value back to the sending federate. The latency is computed by dividing the round-trip time by 2. The throughput program measures RTI performance in terms of the number of attribute update from sending federate to receiving federate in one second. The throughput measurement is processed by two federates as well. One federate is responsible to perform a specified number of attribute updates. The receiving federate sends an acknowledgement back to the sending federate when the specified number of updates is done. In these two benchmarks, we set the size of an attribute value to {1, 4,…, 1024, 4096} Kbytes and all other parameters are set to the default values.

In our architecture, the message is routed with one hop if both the sending and receiving federates belong to the same group. Otherwise, the message is routed with three hops. The latency and throughput results of the DMSO RTI-NG1.3v6 and one-hop and three-hop latency of our architecture are shown in table 1 and table 2 respectively.

From table 1, the one-hop latency of our architecture is a little longer than the latency of DMSO RTI. The three-hop latency of our architecture is about 2.5 times slower than the latency of DMSO RTI. From table 2, the one-hop throughput of our architecture is lower than DMSO RTI in the cases where message size is smaller than 64Kbytes. But when the size of an attribute value is greater than 256Kbytes, the throughput of our design and DMSO RTI are similar and both designs can achieve 40 Mbytes per second. The performance of three-hop throughput is about 28 Mbytes per second when the size of attribute value is greater than 256 Kbytes.

B. Time Advancement Benchmark

The time advancement program measures RTI performance in terms of the rate at which time advance requests are granted. We set the number of federates in a federation execution to {4, 8,…, 256, 512} and all other parameters are set to the default values. The DMSO RTI-NG1.3v6 cannot support more than 64 federates participating in a federation due to its design of joining a federation.

In our RTI implementation, a conservative synchronization protocol [6] is used in time management. We test the total grants per second of our RTI system under 1, 2 and 4 FedServs respectively and each FedServ will manage the same number of federates. The results are shown in figure 4. In figure 4, MF(k) means that there are k FedServs in a federation execution. For the case where number of federates is 4, we can see the total grants of our RTI system using one FedServ is larger than two or four FedServs. But when the number of federates is greater than 32, using four FedServs in a simulation can achieve the better results than one FedServ and two FedServs because the loading of the RTI system is shared among the FedServs. The results show that we can just add more FedServs in the RTI system to support more federates and also preserve the performance.

C. Multicast

The performance of the multicast operation is evaluated by one sending federate and several receiving federates. The sending federate is responsible to send an interaction parameter to each receiving federate and each receiving federate sends the interaction parameter back to the sending federate when receiving it. The elapsed time of the operation is recorded. We set the size of the interaction parameter to 1 Kbytes and 64 Kbytes, and set the number of the receiving federates in a federation execution to {4, 8, …, 128, 256}. The test is executed 100 iterations and the average value is calculated. The performance results are shown in Fig. 5 and Fig 6.

The results in Fig. 5 and Fig. 6 show that as the number of federates increases, the elapsed time of multicast will be further reduced by increasing the number of FedServs. More FedServs involved in the operation of multicast will improve the parallelism in the message transmission from the FedServs to the federates and the performance is better in our Multiple-FedServ architecture.

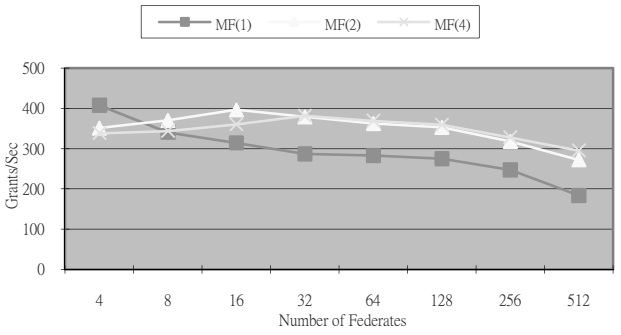

Figure 4. Performance of time advancement benchmark.

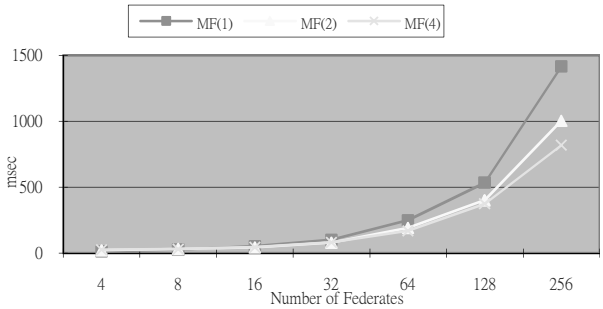

Figure 5. Performance of time advancement benchmark. (message size = 1 Kbytes)

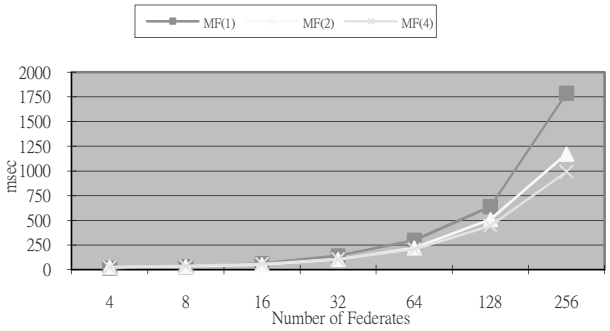

Figure 6. Performance of time advancement benchmark.(message size = 64 Kbytes)

D. Synchronization

The performance of the synchronization operation is evaluated in terms of the elapsed time to synchronize specified number of federates. The elapsed time is calculated starting from the registration of a synchronization point to the point that all federates are synchronized. We set the number of the federates joining the operation of synchronization in a federation execution to {4, 8,…, 128, 256}. The test is executed 100 iterations and the average value is calculated. The results are shown in Fig. 7.

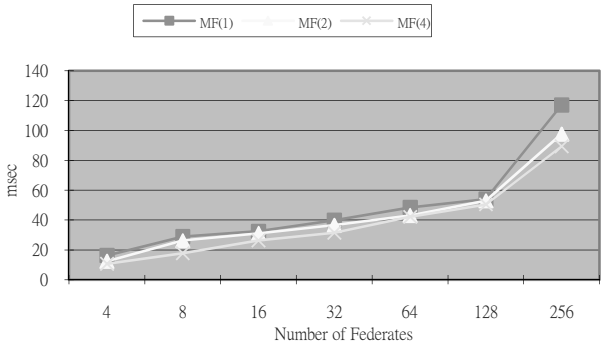

Figure 7. Performance of operation of synchronization.

As depicted in Fig. 7, the elapsed time to synchronize the federates is reduced by increasing the number of the FedServs. When the number of federates is large, the RTI system will perform better by creating more FedServ in the federation execution. The results show that our architecture is beneficial for the operation of synchronization.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we propose a scalable architecture for implementing RTI system by taking advantages both of the centralized and distributed architecture. With a large-scale simulation, the time advancement, operation of multicast and synchronization can benefit from our proposed Multiple-FedServ design. We also provide the connection scheme and synchronization method to further improve the performance of the operations of multicast and synchronization. The experimental results show that the Multiple-FedServ design can scale well and also provides high performance.

In current implementation, the joining operation of federates and FedServs are manipulated by the users. In the future, we will make this operation more flexible and automatically increase/decrease the FedServs by the RTI system according to the loading of the federation execution. Also, we will provide methods to dynamically change the connecting topology at runtime to make the loading of the federation more balanced and make the operation of multicast and synchronization perform better.

ACKNOWLEDGMENT

The work of this paper is partially supported by CHUNG-SHAN Institute of Science & Technology under contract XW98141P and National Science Council under contract 98-2623-E-007-007-D. The authors would like to thank anonymous referees for their comments.

REFERENCES

[1] E. D. Brooks, "The bufferfly barrier", International Journal of Parallel Programming, Volume 15, 1986.

[2] D. Chen, S.J. Turner, W. Cai, "A Framework for Robust HLA-based Distributed Simulations", *Principles of Advanced and Distributed Simulation 20th Workshop*, 2006, pp.183-192.

[3] W. Cai, S. J. Turner, and H. Zhao, "A load management system for running HLA-based distributed simulations over the grid", *Distributed Simulation and Real-Time Applications, Proceedings of the Sixth IEEE International Workshop*, 2002, pp. 7-14.

[4] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration", http://www.globus.org/research/papers/ogsa.pdf, 2002.

[5] R. M. Fujimoto and P. Hoare, "HLA RTI Performance in High Speed LAN Environments", *Fall Simulation Interoperability Workshop*, September 1998.

[6] R.M. Fujimoto, and R.M. Weatherly, "Time Management in the DoD High Level Architecture", *Parallel and Distributed Simulation, PADS 96. Proceedings*, 1996, pp.60-67.

[7] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), IEEE 1516 (HLA Rules), 1516.1 (Interface Specification), 1516.2 (OMT), September, 2000.

[8] T. McLean and R. M. Fujimoto, "The Federated-Simulation Development Kit (FDK), http://www.cc.gatech.edu/computing/pads/fdk.html.

[9] G. Theodoropoulos, Y. Zhang, D. Chen, R. Minson, S. J. Turner, W. Cai, Y. Xie, and B. Logan, "Large Scale Distributed Simulation on the Grid", *Cluster Computing and the Grid Workshops, Sixth IEEE International Symposium*, vol.2, 16-19 May 2006, pp. 63-63.

[10] U.S. Department of Defense Modeling and Simulation Office, HLA RTI-NG 1.3v6.

[11] K. Zajac, M. Bubak, M. Malawski, and P. Sloot, "Towards a grid management system for HLA-based interactive simulations", *Distributed Simulation and Real-Time Applications, Proceedings of Seventh IEEE International Symposium*, 23-25 Oct. 2003, pp. 4-11.

[12] W. Zong, Y. Wang, W. Cai, and S. J. Turner, "Grid Services and Service Discovery for HLA-Based Distributed Simulation", *Distributed Simulation and Real-Time Applications", Eighth IEEE International Symposium*, 21-23 Oct. 2004, pp. 116-124.

Table 1: Latency benchmark (Unit: msec).

	1	4	16	64	256	1024	4096
DMSO	5.767	5.792	6.372	2.225	8.170	29.995	122.721
1 hop	5.481	5.936	5.964	7.360	10.218	35.628	145.181
3 hops	9.048	8.466	10.240	10.085	21.462	70.919	280.306

Table 2: Throughput Benchmark (Unit: MByte/sec).

	1	4	16	64	256	1024	4096
DMSO	6.052	18.503	33.918	38.213	40.184	42.949	43.752
1 hop	0.481	1.914	7.619	30.334	42.218	43.382	44.442
3 hops	0.398	1.612	6.299	19.162	25.757	28.962	29.841