FRAMEWORK FOR ADAPTIVE TIME SYNCHRONIZATION METHOD FOR INTEGRATION OF DISTRIBUTED, HETEROGENEOUS, SUPPLY CHAIN SIMULATIONS

Siddharth Misra, Jayendran Venkateswaran, Young-Jun Son
Systems and Industrial Engineering
The University of Arizona
Tucson, AZ 85721-0020

Abstract
In forming a federation of distributed simulations, their simulation clocks must be synchronized to ensure that events in each simulation are executed correctly, resulting in correct simulation results. We first discuss three time synchronization methods: 1) conservative, 2) optimistic, and 3) scaled real-time. To overcome problems of each of these static approaches, we propose a neural network based adaptive approach, which will react to the dynamic federation environment. In this paper, the framework of the neural network is discussed, and partial experimental results conducted for a distributed supply chain simulation are investigated which will be used as input to train the neural network.

Introduction

Computer simulation modeling has become a standard planning and analysis technique for complex systems. As a system becomes more complex, the complexity of corresponding models increases which in turn places a burden on the computing power of a processor. In the highly competitive environment and market, where product life cycle times are shortened, one critical aspect of conducting simulations is to obtain faster results. Distributed simulation offers a solution to obtain faster results by splitting a simulation model into many modules (sub-models) spread over multiple computers. A major problem associated with this progression is the time synchronization (TS) of these sub-simulations. Several TS mechanisms have been developed, including 1) the conservative method, 2) the optimistic method, and 3) the scaled real time method (Fujimoto, 1998). The conservative method is a strategy where none of local simulations pertaining to a federation advances its time until it ensures that it can not receive any events in the past from other simulations. The time can be advanced in fixed steps or based on the next event time (the nearest event among the simulations) of the federation. In optimistic synchronization, each simulation executes messages (events) received from other simulations whenever they are available under the optimistic assumption of a timestamp ordered execution. Each time when a simulation detects a violation of the timestamp order (i.e., a message time-stamped in the past of its local time arrives), it rolls back to its state (local time) immediately prior to the violation, and the execution resumes (Jefferson and Sowizral, 1985). Under the scaled real time method, the speed of each simulation is factors of the wall clock time (Fujimoto, 1998). For a more detailed discussion on these approaches, refer to Fujimoto (1995, 1999). Each of these mechanisms works well only for a particular type of system, which motivates the need of an adaptive approach.

The advantage of the conservative method is the ease of implementation and reliability (high degree of synchronization). However, this method generates a considerable computational overhead used to ensure proper time advancement. Therefore, the conservative approach is typically used for situations where numerous messages are exchanged among simulations and the sequences of messages are important. The advantage of the optimistic approach is efficient use of computational resources. On the other hand, the disadvantage of this approach is the overwhelmed memory requirements used to store states for a possible rollback and technical difficulties implementing rollbacks (Mattern, 1993; Fujimoto and Hybinette, 1997). Therefore, the optimistic approach is typically used for situations with less number of message transfers. The advantage of the scaled real time approach is considerable reduction of simulation run times. However, it is difficult to determine appropriate scaling factors (degree of scaling) which will provide valid simulation results. Therefore, this method is typically used for situations having a dependable and reliable network, and where missing messages does not have a significant effect in results.

The long-term goal of this research is to enable adaptive identification of the best TS method for dynamically changing system configurations and conditions. In this paper, the framework of a neural network based adaptive choice of TS methods is presented. To this end, we first evaluate the quantitative performance of the different TS approaches. For this purpose, a distributed supply chain simulation is constructed that integrates partners' existing factory-level simulations. In constructing this distributed simulation, different TS approaches have been implemented, and the experimental results obtained are used to train the neural network. The neural networks will be trained to identify the best possible TS method for a given distributed model and conditions. A general framework has been constructed which involves two stages. The first stage analyzes the performance of various TS approaches for the distributed model under study and trains a neural network. The second stage helps in coming up with the best possible TS approach suited to the distributed model given the configuration and goals of the distributed model.

Neural Network Based Adaptive Method

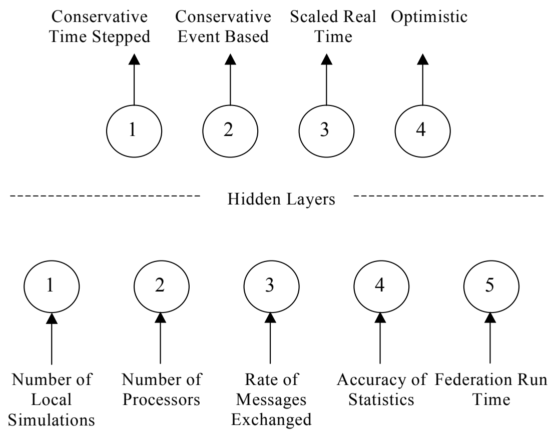
A neural network is used to model complex relationships among critical dependent variables, different TS methods, and performance measures. Whenever there is a need to run a distributed model, the neural network can be used to identify the most efficient TS approach. Also, this framework can be expanded to choose the most efficient TS approach in a dynamic environment. That is, in the course of execution of a distributed model, the current TS approach can be changed using the above framework, as need arises. The overview of the neural network is depicted in Exhibit 1. Exhibit 1 shows the detailed input and output layers except for the internal layers. The input layer has five input nodes, and the output layer has four output nodes. Several parameters must be determined to obtain a complete neural network, including 1) the number of hidden layers, 2) the number of hidden neurons at each layer, and 3) the learning rate. Additionally, all of the weights on the arcs are empirically determined from the training samples.

The TS approaches have been implemented so that we can collect data regarding the performances of each approach after conducting experiments on a common distributed model. The distributed model (federation) mimics a big system which is broken down into many simulation modules (sub-models) to reduce the complexity of the system. The modules forming the distributed model are easy to handle, and the working of the system is easily visible. These modules are then run on different PC's. The data collected from this distributed model is used as training samples. A training sample consists of an input vector and a corresponding output vector. These training samples enable the TS selection method to be adaptive. Once a set of training samples are collected, a neural network will be constructed by obtaining the best weights to minimize the system error E,

$$E = \min \sum_{j=1}^{6} \left(t_{pj} - o_{pj}\right)^2, \qquad (1)$$

where t_{pj} implies the desired output vector for the j^{th} component of the p_{th} training sample and o_{pj} the actual output value generated by an intermediate neural network.

Exhibit 1. Overview of neural network



To train the neural network offline, certain variables (see Exhibit 1) have been identified, which will be used as the inputs to the neural network. These variables are further classified as fixed configuration variables and performance measures. The fixed configuration variables, such as 1) the number of local simulations, 2) the number of processors, and 3) the rates of messages exchanged, are decided before the simulation runs are carried out. On the other hand, the performance measures, such as 1) the accuracy of statistics and 2) the federation speed, are available only after simulations are run. It is noted that the performance measures depend on both the type of synchronization method and the fixed configuration variables. The first input, the number of local simulations is the measure of complexity of the single simulation. In general, less number of simulations means that each local simulation is more complex. The number of simulations can take on values from 2 to N. The second input, the number of processors, is the number of PCs on which the simulations are run. This input is less than or equal to the number of simulations. The third input, the rate of messages exchanged, is a measure of how busy the network is. It can be of either of two levels, +1 representing a busy/high intensity network and -1 representing a low intensity network. The fourth input, the accuracy of the statistics, is the percentage difference between the statistics obtained from running the distributed model and the statistics obtained from running a single (as opposed to distributed) base model. Details about the base model are presented in the Experiments Section. The fifth input, the federation run time, is the actual time taken to complete a distributed simulation run in the wall clock time. In general, the lesser the time taken to complete the simulation the better. A data set for training the neural network will represent all the five inputs discussed here.

The output of the neural networks is one of the four TS approaches (refer Exhibit 1). Differences between the conservative time-stepped and the conservative event based approaches will be explained in Implementation Platform section. Each data set is mapped to a particular output based on the experiments carried out after implementing each TS approach. The value for each of the outputs is a binary value (0 or 1) indicating the usage of that particular TS approach. The output can be represented as a set: (x, y, z, w) where x represents conservative time stepped, y represents conservative event based, z represents scaled real time, and w represents optimistic approach. While training, only one of these four values will be 1, and the rest will be 0. That is, only 4 outputs sequences are possible: (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), and (0, 0, 0, 1).

Exhibit 2 shows example data sets used for training the neural network. In Exhibit 2, the first data set used for training the neural network is interpreted as follows: there are 4 simulations running on 4 processors. The distributed model is run in a high intensity network. The accuracy (percentage difference) of statistics is obtained from experiments conducted and is found to be 5.5% of the actual value. The federation run time is observed to be 0.92 minutes. The TS approach used (output) is the conservative time stepped approach. Training of neural network with extensive data sets is being conducted and is not included in this paper.

Exhibit 2: Data sets for training neural networks

No. of Federates	No. of Processors	Accuracy of Statistics	Rate of Messages Exchanged	Federation Run Time	Output
4	4	5.5%	+1	0.92 minutes	Time Stepped
4	3	19%	-1	6.02 minutes	Event Based
4	4	8%	+1	0.15 minutes	Scaled Real Time
3	3	10.5%	-1	0.50 minutes	Optimistic

Once the neural network is trained offline, it can be applied to a distributed simulation online. In the operation stage, each node of the neural network shown in Exhibit 1 has slightly different context compared with the training stage. In the operation stage, the input nodes are classified into two (Exhibit 1): *desired* configuration inputs (inputs 1 and 2) and *desired* goal inputs (inputs 3, 4 and 5). The desired configuration inputs are the number of simulations and the number of processors of the distributed model. Their usage is the same as when the neural network is trained. The desired goal inputs are the desired accuracy of statistics, the desired federation run time, and the desired rate of messages exchanged. These are based on the users' expectation from the distributed model. Hence, the accuracy of statistics is the *expected* percentage difference between the statistics obtained from the distributed model and the single base model; the rate of messages exchanged is the *expected* measure of how busy the network is; the federation run time is the time *available* with the user to run the distributed simulation. The output of this neural network will be one of the different TS approaches. The outputs are anticipated not to be exactly 0s and 1s. Instead, they will be fractions between 0 and 1. An example data set used in the operation stage of the neural network is depicted in Exhibit 3. The inputs given by the user specify the desired configuration as 4 simulations run on 4 processors in a high intensity network with the desired goal of the distributed model to be within 5% of actual statistics with the desired federation run time of 1 minute. For those inputs, the output of the neural network can be (0.91, 0.3, 0.79, 0.53). This output suggests the best TS approach for the specific configuration is the conservative time stepped approach followed by the scaled real time approach.

Exhibit 3: Example data set used in the operation stage of neural network

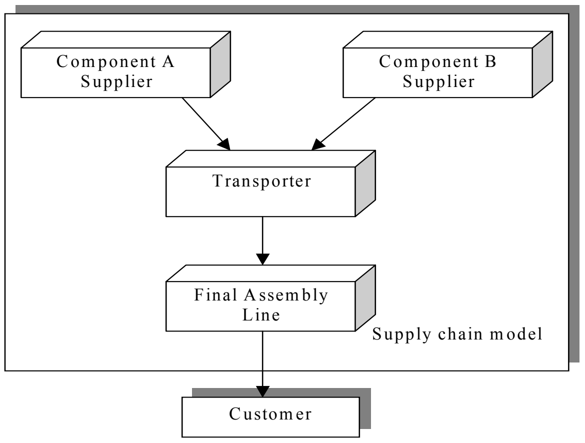
No. of Federates	No. of Processors	Accuracy of Statistics	Rate of Messages Exchanged	Federation Run Time	Output
4	4	5 %	+1	1 minute	Time Synchronization Approach

Prototype Supply Chain

The prototype supply chain considered in this research to evaluate the performance of different TS approaches is a pull system composed of an assembly plant, two suppliers, and a transportation system (see Exhibit 4). The Assembly plant, which acts like the Original Equipment Manufacturer (OEM), produces the final products from the parts produced by

Supplier A and/or Supplier B. The transporter moves the parts from the suppliers' plants to the assembly plant. The assembly plant maintains a buffer stock of all the components. When the stock of any of the components falls below a prescribed threshold, a purchase order is issued to the corresponding supplier. The suppliers also maintain a minimum level of stock to ensure that the assembly plant always gets components as quickly as possible.

Exhibit 4. Prototype supply chain

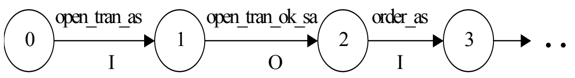


The behavior of the supply chain and the messages exchanged between the members remains the same as described by Venkateswaran et al. (2001). It is noted that this particular supply chain configuration is adopted for illustration purposes, and the proposed method in this paper can be applied to other complex systems such as manufacturing, supply-chain (different configurations), telecommunication and computer network systems.

Communication Formalism and Protocol

Local simulations in a federation exchange information with other simulations or the federation manager through messaging. Two major purposes of these interactions are: 1) notification of timestamps used for the synchronization of the federation time and 2) delivery of an event that initiates appropriate actions (e.g., creation or disposal of entities) in a simulation. In the case of time notification, only the time stamp of each message is important. However, in the case of delivery of an event, the semantics as well as the time stamp of each message are important. In this research, each simulation is equipped with a formal behavior model that enables coordinative interactions with other simulations. For this purpose, a modified deterministic finite automaton (MDFA) similar to a Mealy machine (Hopcroft and Ullman, 1979) has been employed. A modified deterministic finite automaton, ***M***, is defined formally as the octuple, $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{q}_0, \boldsymbol{F}, \boldsymbol{\Sigma}, \boldsymbol{ACT}, \boldsymbol{P}, \delta, \gamma)$, where ***Q*** is a finite set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states. ***Σ*** is a finite set of *federate events* and serves as the input alphabet. Additionally, ***Σ*** is partitioned into a set of input messages ($\boldsymbol{\Sigma_I}$), a set of output messages ($\boldsymbol{\Sigma_O}$), a set of federate tasks ($\boldsymbol{\Sigma_T}$), and a set of observation tasks ($\boldsymbol{\Sigma_{Ob}}$). ***ACT*** is a finite set of *federate actions*, where each $\alpha \in \boldsymbol{ACT}$ is a procedure that performs some action (e.g., creation and disposal of entities) in the federate. $\delta: \boldsymbol{Q} \times \boldsymbol{\Sigma} \rightarrow \boldsymbol{Q}$ is a state transition function. Similarly, $\gamma: \boldsymbol{Q} \times (\boldsymbol{\Sigma_O} \cup \boldsymbol{\Sigma_T}) \rightarrow \boldsymbol{ACT}$ is a federate action transition function. This communication protocol has been implemented using the functions provided in the High Level Architecture-Run Time Infrastructure (HLA–RTI) (Venkateswaran et al., 2001). The benefit of using a formal model is that it forms a basis for software development. A simple Deterministic Finite State Automata (DFSA) for a supplier is shown in Exhibit 5. The "I", "O" and "T" denote the incoming messages, outgoing messages and the tasks carried out, respectively. In addition, "_as" messages go from the assembler to a supplier; "_sa" messages go from the supplier to the assembler; and, "_st" messages go from a supplier to transporter. In the sequence shown in Exhibit 5, the assembler initiates the transaction with the *open_tran_as* message and the supplier replies with the *open_tran_ok_sa* agreement message. After the successful generation of an order, the assembler sends the *order_as* message. Due to limited space, the other messages (events) are not explicitly explained.

Exhibit 5. Finite state automata for supplier



Implementation Platform

The High Level Architecture (HLA) has been developed by Defense Modeling and Simulation Office (DMSO) to provide a consistent approach for integrating distributed, heterogeneous, and defense simulations. Run Time Infrastructure (RTI) software, which implements the rules and specifications of HLA, provides methods which can be called and used by individual simulations. The direct interaction of the simulations with the RTI is quite complex. To respond this problem, NIST has developed the Distributed Manufacturing Simulation (DMS) Adaptor, a software interface, to provide mechanisms for distributed simulation similar to those provided by the HLA RTI, but with a level of complexity that is manageable by the development resources available in the manufacturing community (Riddick and McLean, 2000). In the experiments conducted in this research, all the local simulations are built in Arena^{TM}, which have been integrated into a federation based on the HLA/RTI and the DMS adaptor.

The conservative method has been implemented using two ways to advance time in a simulation: 1) the time stepped approach and 2) the event based approach.

1. Time Stepped: In the time-stepped approach, each simulation advances its time by a fixed step size. Appropriate step sizes depend on the federation conditions. In this paper, multiples of ΔT, the average inter-event time calculated from the base model, are used as step sizes. A generic algorithm (pseudo code) to calculate the ΔT has been developed and is shown as follows:

```
One entity is created at time zero (this entity is used to calculate inter-event time)
Temp_time = 0
While TNOW ≤ simulation termination time
    While TNOW > Temp_time
        <<Do Nothing>>
    End while
    Collect an inter-event time (TNOW – Temp_time)
    Temp_time = TNOW
End while
```

The DMS adaptor has a constraint in the way it handled the time-stepped approach. Time can only be incremented in the multiples of the value 10. To overcome this constraint, a scaling mechanism has been added in the model code. The scaling mechanism reports a simulation advance request in a magnified scale while each simulation is still progressing in the small-scale quantities. When being fed back to the simulation, the time is scaled down by the factor it was scaled up.

2. Event Based: In the event based conservative approach, each simulation advances its time by going to its next event in the federation having the smallest logical time. In this case, each simulation announces its next event time to the RTI, and the RTI allows its advance to that event depending on the request of other simulations. Therefore, whenever an event is noticed, the step size will be changed to the difference between the next event time and the present simulation time.

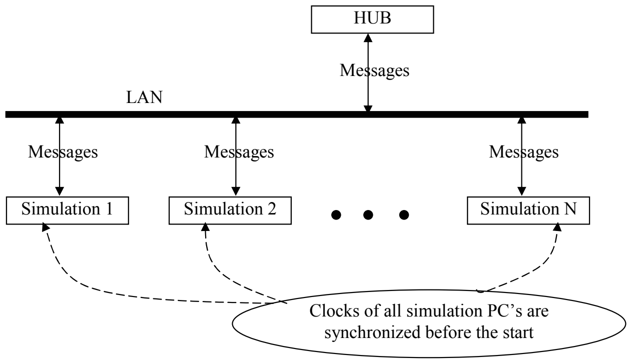
The scaled real time approach has been implemented using client/server applications. In the real time approach, the simulations are executed synchronously with the real time (wall clock). This means that 1 minute of the simulation time corresponds to 1 minute of the wall clock time. If two simulations begin at the exactly same time, both simulations times will be exactly same as they correspond with the real time. Assuming negligible network latency, and other system delays, the messages sent by one simulation are assumed to reach the other simulation instantaneously. The same logic can be applied to N simulations running in parallel with the exactly same simulation times. In the case of the scaled real time, 1 minute of the wall clock time is scaled down so that the simulation time is less than 1 minute. That is 1 minute of the wall clock time can be scaled to represent 0.5 minute of the simulation time (scaling factor of 2 in this case). However, this truly distributed arrangement requires that each simulation keep track of information about all the other simulations. Hence, to facilitate easier communication between simulations, a central server called 'Hub' is conceived, which stores the information pertaining to all the simulations in the distributed network. In this implementation, the Hub has been used in place of the HLA/RTI to reduce the overhead associated with the HLA/RTI. The architecture of the system is depicted in Exhibit 6. Multiple simulations can be run in parallel with the same simulation time. Each simulation sends all its messages to the 'Hub'. The 'Hub' then redirects the message to the respective receiver. The clocks on all the simulation PC's are synchronized before the start of the simulations using Dimension 4$^{®}$ software. It is important to note that following points:

- The Dimension 4$^{®}$ software handles the coordination of the simulation PC's time before the start of the simulation run. Since the simulation time is coordinated with the system clock, the advancement of time in all simulations is assumed to be the same.
- The work of the 'Hub' is restricted in redirection of messages from one simulation to another. It does not perform any time management or object management.

The 'Hub' is coded in Java using socket programming. The 'Hub' once started, will wait for new client connections from the simulations. Once a new connection is accepted, a separate process (thread) is spawn, which listens for messages from the client simulations. When all the required simulations are connected, the 'Hub' sends a message to each of the client simulations stating the real time at which the simulations are to start execution. Since the system time of all the simulation PC's are synchronized using Dimension 4$^{®}$ software, all simulations will begin at the exactly same instant. The simulations are then run. Messages are received by the 'Hub' along with the destination information.

The 'Hub' then redirects the message to that particular client simulation.

Exhibit 6. Architecture of real time/time scaled distributed simulation



The optimistic approach has not been implemented yet. The saving and restoring of states is of prime importance for implementing the optimistic approach. For a possible roll back, each simulation has to save past states. The foremost problem encountered is that Arena^{TM} does not provide functions to save and restore states. In fact, there is limited number of software which provides the capability to save and restore states. Some software which claims to provide these functions have been evaluated, however, they required users to understand the internal structure of data used for developing the software. As mentioned earlier, one of the major disadvantages of the optimistic approach is difficulty associated with implementing it. Another issues involved in implementing the optimistic approach are answering the following questions: 1) how often should the states be saved and 2) how far the simulations should be allowed to run ahead of other simulations. All the challenges discussed here will be continuously studied.

Experiments

Experiments have been conducted to study the feasibility of the proposed framework to evaluate the best TS approach for a particular network and the given prototype supply chain model (see Exhibit 4). The data has been collected for a base model as well as the distributed model of the supply chain. The base model represents a monolithic simulation model containing all the players of the chain. The base model is considered as the reference model for the statistics, i.e., the statistics obtained from the base model are considered as the true results; this assumption was necessary since there is no such real system. The statistics of the distributed model with different configurations (including different TS methods) are compared with the true values (base model statistics).

The simulation models have been distributed over four PC's connected through a LAN network. All the computers on which the experiments were conducted are Pentium^{TM} 4 machines. The experiments have been conducted during a low traffic period in the LAN. The experiments discussed in this paper are classified in two categories: 1) conservative runs (A) time stepped and (B) event based and 2) real time scaled runs. After evaluating the outputs of the conservative runs, the time step with minimum error in statistics as compared to true values is chosen to represent the best time step. The data for this time step will be used to train the neural network in the future. Similarly, after analyzing all the real time scaled runs, the one with the minimum execution time with insignificant discrepancy in statistics will be used to train the neural network in the future.

Results

The results for the base model and the distributed model are presented here. The utilization (%) of the assembler and transporters from different models are presented in Exhibits 7 through 13. In Exhibits 7, 9, and 11, the utilization of assembler and transporters are plotted against the step size for incrementing time (time step) whereas in Exhibits 8, 10, and 12, the utilization of assembler and transporters are plotted against the real time scaling factor. In each exhibit, the thick line is the base model statistics; the dotted lines show the conservative time stepped statistics; and the curved dashed lines show the scaled real time statistics.

Exhibit 7. Utilization of assembler for conservative time stepped approach

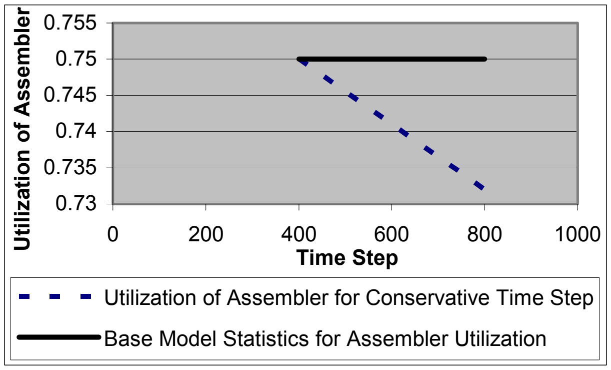


Exhibit 8. Utilization of assembler for scaled real time approach

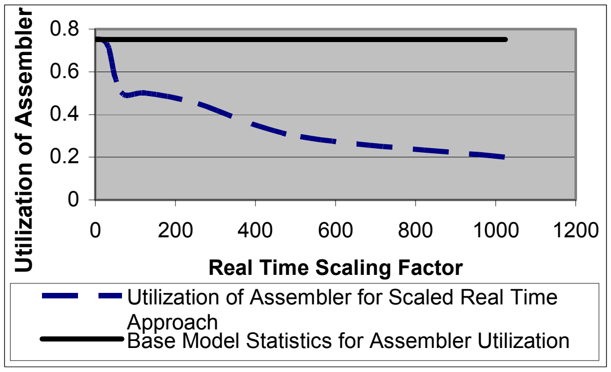


Exhibit 9. Utilization of transporter 1 for conservative time stepped approach

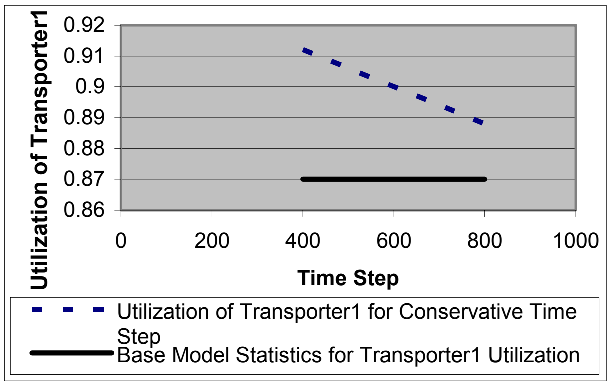


Exhibit 10. Utilization of transporter 1 for scaled real time approach

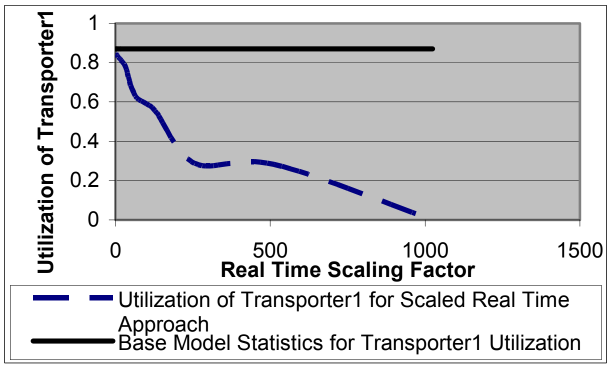


Exhibit 11. Utilization of transporter 2 for conservative time stepped approach

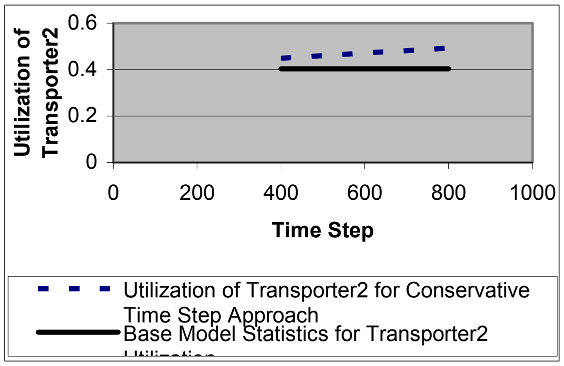


Exhibit 12. Utilization of transporter 2 for scaled real time approach

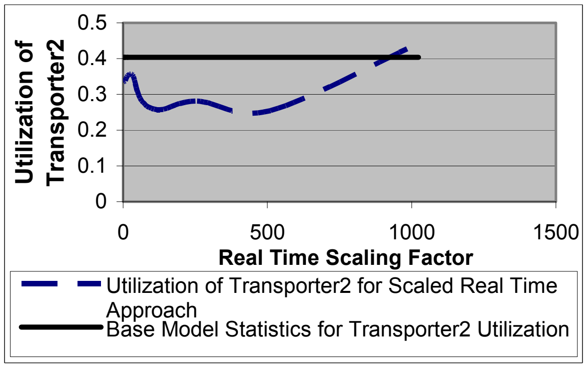


Exhibit 13. Statistics for event based approach

	Base Model Statistics	Event Based
Utilization(Assembler)	0.75	0.45
Utilization(Transport1)	0.87	0.7456
Utilization(Transporter2)	0.40333	0.39493

In Exhibit 7, it is shown that the utilization for the assembler coincides with the base model statistics for time increment of the distributed model based on a fixed step size of 400, however, a slight discrepancy is observed as the fixed step size for time increment of the distributed model is increased to 800. In Exhibit 9, the utilization of the transporter 1 is observed to be close to the base model statistics. Similarly, in Exhibit 11, the utilization of the transporter 2 is observed to be close to the base model statistics. In Exhibits 8, 10, and 12, the utilization for the assembler, transporter 1, and transporter 2, appears to be close to the base model statistics up to the real time factor of 32, however, a significant discrepancy is observed when the real time factor is beyond 32. Some of the statistics obtained from the event based approach shown in Exhibit 13 illustrate a significant deviation from the base model statistics.

In general, we have found that the results show little or no discrepancy from the base model results for the time-stepped approach. Some of the results for the event based approach show a significant discrepancy from the base model results. It is also seen for the scaled real time approach when the real time factors are greater than 32 variations are observed in the different statistics. Also, it is observed that there is a discrepancy in the exchange of messages, i.e., not all messages are exchanged.

In this paper, partial experimental results conducted for a distributed supply chain simulation have been presented. On the basis of these experiments, extensive and comprehensive experiments will be conducted to generate input data to train the neural network.

Conclusion and Future Work

A framework has been developed which will identify the best time synchronization (TS) approach for the prototype system. In this paper, the neural network based adaptive approach has been presented. In the future, the neural network will be trained off-line from the data collected from running the common distributed model with the time stepped, event based, optimistic and scaled real time TS approaches. For illustration purposes, a limited set of system configurations have been used in this paper.

Currently, we have implemented a conservative time stepped, conservative event based, and scaled real time approaches for TS of a distributed simulation model and compared the results with those (reference values) of a single base simulation model. We have found that the results for the time-stepped approach are closer to the reference values than the event based approach. The results of the scaled real time models have been found to be close to the reference values when the real time scaling factors are smaller than 32. The optimistic approach has been proved to be difficult to implement as most of the software packages do not provide the capability to save and restore states. In the future, we will completely implement the optimistic approach.

The proposed framework will be generalized so that it works for various system configurations and conditions. All the simulations in the experiments presented in this paper are built in Arena^{TM}. In the future, we will use different simulation languages\software to introduce another experimental factor, heterogeneity in the federation.

References

Fujimoto, Richard M, "Time Management in High Level Architecture", *Simulation*, Vol. 71, No. 6 (1998), pp. 388-400.

Jefferson, David and Sowizral, H., "Fast concurrent simulation using the Time Warp mechanism", *Proceedings of the SCS Multiconference on Distributed simulation*, Vol. 15, No. 2, (January 24-26, 1985), pp. 63-69.

Fujimoto, Richard M, "Parallel and Distributed Simulation", *Proceedings of the 1995 WSC,* (December 1995), pp. 118-125.

Fujimoto, Richard M, *Parallel and Distributed Simulation Systems*, Wiley Publishers (1999).

Mattern, F., "Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation", *Journal of Parallel and Distributed Computing*, Vol. 18, No. 4, (1993), pp. 423-434.

Fujimoto, Richard M, Hybinette, M., "Computing Global Virtual Time in Shared-Memory Multiprocessors", *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 2, (October 1997), pp. 425–446.

Venkateswaran, J., Yaseen, K., and Son, Y., "Distributed Simulation: An Enabling Technology for the Evaluation of Virtual Enterprises", *Proceedings of the 2001 Winter Simulation Conference,* (December 2001), pp. 856-862.

Hopcroft J.E. and Ullman J.D., *Introduction to automata theory, languages and computation*, Addison-Wesley (1979).

Riddick, F., Mclean, C., "The IMS Mission Architecture for Distributed Manufacturing Simulation", *Proceedings of the 2000 Winter Simulation Conference*, (December 10-13, 2000), pp. 1539-1548.

About the Authors

Siddharth Misra is a Master's student at University of Arizona in Tucson in Industrial Engineering. He holds a B.E. degree in Mechanical Engineering from the University of Pune. He is currently a research assistant at the Computer Integrated Manufacturing Lab of University of Arizona. His current research focuses on time synchronization mechanisms in distributed simulations.

Jayendran Venkateswaran is a Doctoral student at University of Arizona in Tucson in Systems and Industrial Engineering. He holds a M.S. degree in Industrial Engineering. He is currently a research assistant at the Computer Integrated Manufacturing Lab of University of Arizona. His current research focuses on system dynamics, distributed simulation and supply chain management.

Dr. Young Jun Son received his Ph.D. from Pennsylvania State University in Industrial and Manufacturing Engineering. He holds an M.S. degree in Industrial and Manufacturing Engineering from the Pennsylvania State University and a B.S. degree in Industrial Engineering from POSTECH, Korea. He is currently an Assistant Professor in Systems and Industrial Engineering and the director of the Computer Integrated Manufacturing Lab at The University of Arizona. His research work involves distributed and hybrid simulation for analysis and control of automated manufacturing system and integrated supply-chain. He is an associate editor of the International Journal of Modeling and Simulation and a professional member of ASME, IEEE, IIE, INFORMS, and SME.