Distributed Virtual Simulation Characterization for Performance and Scalability Estimation

Douglas D. Hodson
ASC Capabilities Integration Directorate
WPAFB, OH
douglas.hodson@wpafb.af.mil

Rusty O. Baldwin
Air Force Institute of Technology
WPAFB, OH
rusty.baldwin@afit.edu

William K. McQuay
AFRL Collaborative Simulation Technology
WPAFB, OH
william.mcquay@wpafb.af.mil

ABSTRACT

The field of distributed virtual simulation has typically been associated with training human operators. While training is still a principle design goal, large scale distributed virtual simulations are increasingly being used to analyze assets within the simulation itself. In other words, the trend is to use distributed virtual simulations for the purpose of solving more analytic simulation problems. This paradigm shift requires more formal methods to ensure that requirements from both human participants and analytic models are being satisfied.

Considerable research has been done to capture human interaction requirements which determine the virtual environment that needs to be created, but little research has been done to characterize distributed virtual simulations in general, especially when analytic model requirements need to be considered. This paper will present a framework to characterize distributed virtual simulations in terms of a temporal data consistency model so that the performance and scalability of system designs can be estimated. It also presents initial performance and scalability results for a DIS-based simulation system.

ABOUT THE AUTHORS

Douglas D. Hodson is an Electrical Engineer at the Simulation and Analysis Facility, Wright Patterson Air Force Base AFB, OH. He is the technical lead of the Extensible Architecture for the Analysis and Generation of Linked Simulations (EAAGLES) software framework. He received a B.S. in Physics from Wright State University in 1985, and both an M.S. in Electro-Optics in 1987 and an M.B.A. in 1999 from the University of Dayton. He is also a graduate student and Adjunct Instructor at the Air Force Institute of Technology working towards his Ph.D. in Computer Engineering.

Rusty O. Baldwin is an Associate Professor of Computer Engineering in the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH. He received a B.S. in Electrical Engineering (cum laude) from New Mexico State University in 1987, an M.S. in Computer Engineering from the Air Force Institute of Technology in 1992, and a Ph.D. in Electrical Engineering from Virginia Polytechnic Institute and State University in 1999. He served 23 years in the United States Air Force. He is a registered Professional Engineer in Ohio and a member of Eta Kappa Nu, and a Senior Member of IEEE. His research interests include computer communication networks, embedded and wireless networking, information assurance, and reconfigurable computing systems.

William K. McQuay is Technical Advisor, Collaborative Simulation Technology and Applications Branch, Information Systems Division, Information Directorate, Air Force Research Laboratory, Wright Research Site. He is an internationally recognized expert in modeling and simulation and distributed collaborative environment technology. He has over 38 years experience in research for advanced simulation technology and the development and utilization of digital simulation models for the Air Force research, development, acquisition, and test and evaluation process. In 2003, Mr. McQuay was selected as an AFRL Fellow for life time achievements and significant advancements in modeling and simulation technologies and collaborative sciences. He has a distinguished record of contributions in modeling and simulation going back to the early days of computer based simulation of electronic phenomena through today's use of simulation to support operational decision makers. Mr. McQuay received a B.S (Mathematics) from Towson University, a Certificate in Meteorology from Texas A&M University, a Master of Applied Science (Operations Research) from Southern Methodist University and a Master of Science in Engineering (Computer Science) from The Johns Hopkins University.

Distributed Virtual Simulation Characterization for Performance and Scalability Estimation

Douglas D. Hodson
ASC Capabilities Integration Directorate
WPAFB, OH
douglas.hodson@wpafb.af.mil

Rusty O. Baldwin
Air Force Institute of Technology
WPAFB, OH
rusty.baldwin@afit.edu

William K. McQuay
AFRL Collaborative Simulation Technology
WPAFB, OH
william.mcquay@wpafb.af.mil

INTRODUCTION

The field of distributed virtual simulation has typically been associated with training human operators. While training is still a principle design goal, large scale distributed virtual simulations are increasingly being used to analyze assets within the simulation itself. In other words, the trend is to use distributed virtual simulations for the purpose of solving more analytic simulation problems. This paradigm shift requires more formal methods to ensure that timely requirements imposed by humans and hardware-in-the-loop, and analytic models are being satisfied.

Considerable research has been done to capture human interaction requirements which determine the virtual environment that needs to be created, but little research has been done to characterize distributed virtual simulations in general, especially when analytic model requirements need to be considered.

It is known that typical distributed virtual simulation systems are implemented as real-time systems that, when distributed, run asynchronously. Furthermore, simulation state space data is not consistent across the nodes that constitute the simulation environment. This inconsistency can be represented by a temporal data consistency model so that estimates of performance and scalability of distributed virtual simulation systems can be evaluated.

BACKGROUND

A distributed virtual simulation is a software system that creates an environment where multiple users participating in the simulation can interact with each other in real-time, even though users may be located around the world. In this context, real-time, means that time with respect to the simulation advances and is synchronized with the wall clock. Distributed in this context refers to heterogeneous computers physically located in different geographic locations connected by a network.

Participants interacting with the simulated environment could include pilots flying fighter aircraft, operators controlling the operation of an early warning radar systems in an Integrated Air Defense System (IADS), or even a person playing a networked game where the objective of the simulation (or game) is less about representing an accurate picture of the real world and more about providing an exciting “virtual world.”

Excluding multi-player gaming, the Department of Defense (DoD) is the largest developer of these types of systems (Singhal and Zyda, 1999) and invests a significant amount of money into large scale geographically distributed virtual simulation systems.

The principle goal in most distributed virtual simulations is to achieve a “sufficiently realistic” representation of an actual or imagined system as perceived by the participants embedded into the environment (Fujimoto, 2000). What “sufficiently realistic” means depends on the underlying requirements for the system.

In many cases, requirements tend to focus on the training activities of participants. To improve the performance of the system, the “state” (i.e., data) of the simulation is replicated in a way that limits network activity and reduces the consistency of the data. Purposely allowing inconsistencies to enhance scalability is sometimes called a “dynamic shared state” (Singhal and Zyda, 1999). This inconsistency allows the system to scale more so a larger number of entities can be represented and included within the simulation itself.

On a completely different track is the development of “parallel analytic simulations” (Fujimoto, 2000). Whereas distributed virtual simulations connect heterogeneous simulators, parallel analytic simulators are designed around a set of homogeneous computers tightly connected through high speed, low latency interconnects or networks. This realm of computing uses multiprocessor computers or clusters of computers to execute a simulation.

Parallel analytic simulation typically perform a quantitative analysis of complex systems. If a person is participating, they simply are an external observer of the system (Fujimoto, 2000). Furthermore, inconsistency of simulation state is not tolerated. Every aspect of the environment has to be absolutely accurate in its minutest detail. It is useful to distinguish between *analytic* and *virtual* simulations because they have different objectives, leading to different requirements and constraints.

Having stated this clear distinction does not take into account that simulation engineers routinely apply systems designed for one domain in the other to conduct simulation studies. For example, the best behavioral model of a pilot is a real pilot. In this case, no computer algorithm can match the real thing. So for some analytical studies in which the system under test involves a person, the constructs used to build virtual simulations might be interleaved with constructs used to build purely analytic simulations.

This research examines the performance and scalability of distributed simulations where requirements can be stated in the form of data consistency. In other words, requirements associated with providing a "sufficiently realistic" environment for a person to participate (such as a visual system requirement) can be quite different than analytic requirements associated with a simulation model.

The strategy for approaching this problem is to characterize the system in terms of a consistency model. In this case, a temporal data consistency model has been identified and is mapped to the problem domain.

Research into this area will facilitate the creation of distributed virtual simulation systems that account for the requirements imposed by human participation, hardware-in-the-loop interaction and analytic models. This is the first research effort that effectively connects two very different worlds of simulation and situates them on a "sliding scale" in which temporal requirements are very different.

ABOUT TIME & STATE

There are several ways to model time in a simulation as shown in Figure 1. The two most common methods include advancing time through a series of events (event-driven) or through a number of fixed steps (time-stepped) with each method offering its own advantages and disadvantages. A continuous simulation changes its state variables continuously with respect to time.

Typically, distributed virtual simulations advance time through a series of time-steps (commonly referred to as delta time). Time-stepped simulators can be designed and more naturally fit the software paradigms associated with real-time systems.

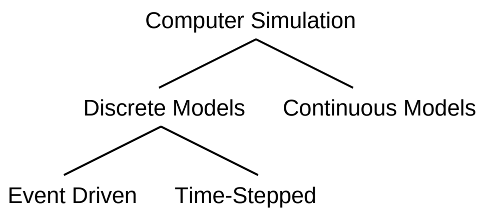

Figure 1: Simulation Time (Fujimoto, 2000)

The system state of a simulation is a collection of state variables necessary to describe the system according to the simulation objective and measured performance metrics at a particular time (Law and Kelton, 2000). For example, the state or position of an object moving through space might be described by an (i,j,k) position vector that changes as time advances. The entire state of a simulation is called its state space.

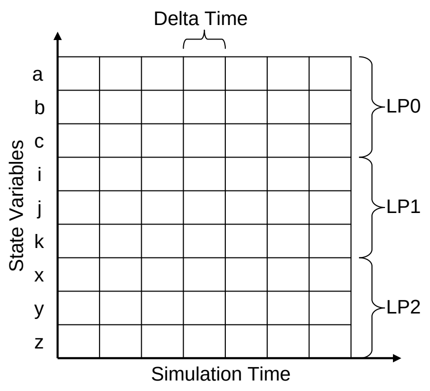

Figure 2: Synchronous State Space Diagram

Distributing a time-stepped simulation across multiple computers involves dividing and partitioning the state space into several Logical Processes (LP). Each logical process is responsible for maintaining its own state. If a synchronization mechanism advances time in a coordinated manner, the resulting state space of the entire distributed simulation state space can be viewed as shown in Figure 2. In this diagram three logical processes (LP0, LP1, LP2) each contain three state space variables of interest.

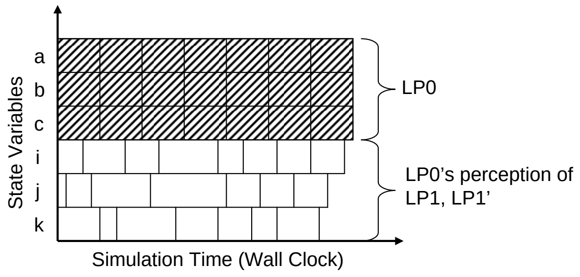

Figure 3: LP0's Perception of LP1 State Space

In reality, the state space of a distributed virtual simulation is purposely allowed to become inconsistent in order to promote the scalability of the system. Because of this inconsistency, each simulator has a different perception of the state space information (variables) as shown in Figure 3.

INTEROPERABILITY

Communication between logical processes (simulators) is facilitated by a number of defined protocol or standards. Two well defined standards are DIS and HLA. TENA is also emerging as a standard in certain application areas. Regardless of the standard used, state information from logical processes are replicated throughout the simulation system and in order to facilitate scalability, the state space of the simulation itself is purposely allowed to become inconsistent. Because the simulation system is executing in real-time and synchronized to a wall clock, replicated state spaces contain data that is older or "aged" compared to the most current value. In fact, a DIS compliant simulation almost always works with aged data. For example, the position of dynamic entities moving in the simulation are rarely consistent with the correct position.

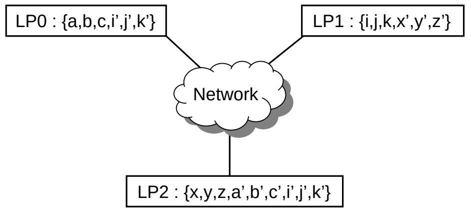

Figure 4: Asynchronous Distributed Simulation

As is shown in Figure 4, logical processes might need to replicate some of the state space that it does not manage using an interoperability protocol such as DIS, HLA or other communication methods. This is especially true when a human operator is inserted into the virtual environment. Therefore, some of the state space represented at any node, at any given point in time, will not necessarily be consistent with the true state.

Depending upon requirements, this inconsistency might be acceptable. For example, the DIS standard specifies that as long as entity position (state) updates are received within 300ms of when they are sent, it is considered acceptable. To reduce network traffic, updates are not sent on a regular periodic basis; they are only sent when certain error thresholds are exceeded. The old or "aged" data is considered good enough and is used to estimate current position until the next update is received.

The level of inconsistency tolerated is based upon a number of factors, including the accuracy of the estimation that can be made with older data, and secondly, the requirement of any virtual environment suitable for the human to participate.

TEMPORAL CONSISTENCY

One of the first steps in characterizing a distributed virtual simulation is to identify a proper consistency model. A consistency model is just that, a model of how memory behaves. It is, in the language of Tanenbaum (Tanenbaum, 1995), a contract between software and memory. A wide spectrum of contracts have been defined, each with a different level of consistency.

It is known that weaker consistency models have a positive effect on the performance of parallel shared memory machines and the benefits increase as memory latency increases (Mosberger, 1993). It is also known that in loosely-coupled systems, such as distributed computers connected through a network, the intermachine message latency is considered large (Tanenbaum, 1995).

For this domain a temporal consistency model has been identified. Temporal consistency is defined in terms of the "age" and "dispersion" of data (Song and Liu, 1995). It is concerned with the time characteristics of data objects read and written by tasks.

The age of a data item x can be expressed by an aging function $a_t(x)$. The dispersion of two data objects is the difference between their ages. For example, let $a_t(x)$ and $a_t(y)$ be the ages of the objects x and y at t, then the dispersion $d_t(x, y)$ would be

$$d_t(x, y) = |a_t(x) - a_t(y)| .$$

Given a set Q of images and derived objects, Q is absolutely temporally consistent at time t if

$$a_t(x) \leq A \text{ where } (A \geq 0)$$

for every x in Q, where A is an absolute threshold (Song and Liu, 1995). Q is relatively temporally consistent at time t if

$$d_t(x, y) \leq R \text{ where } (R \geq 0)$$

for every two objects x and y in Q, where R is a relative threshold (Song and Liu, 1995). A set of data objects is temporally inconsistent if the objects are either absolutely or relatively temporally inconsistent.

The thresholds A and R reflect the temporal requirements of the application. That is, how current and close in age the data must be for the results of computations based on them to be considered correct.

PERFORMANCE ANALYSIS

Assuming temporal model threshold requirements A and R are defined for a system, there needs to be a way to evaluate the overall performance of a system design. Performance in this context quantifies how well a system design meets its temporal requirements. In other words, given temporal thresholds, or bounds, how well does the dynamic system stay within those bounds?

There are three techniques to evaluate the performance of a system: measurement, simulation, and analytic modeling (Jain, 1991). Direct measurement could be done, but in this domain would be rather expensive depending upon a number of factors and requirements. This is especially true with large scale distributed virtual simulation systems.

Analytic modeling is another approach. Certainly "back of the envelope" estimates can be calculated, using network bandwidth and latency values, message transmission rates, and so on. But analytic models tend to be simplifications of a system. As such, important characteristics about the system might be difficult to quantify such as the asynchronous nature of distributed virtual simulations. In fact, asynchronous real-time systems are quite difficult to analyze (Liu, 2000).

This research uses simulation to estimate the performance of a new system design. Specifically, this research uses the sound mathematical foundation of Petri net theory to estimate system performance through simulation.

PETRI NETS

Petri nets are a graphical and mathematical tool to model, analyze and simulate discrete-event systems and model discrete distributed systems (Peterson, 1977). They originated from the doctoral dissertation of Carl Adam Petri in 1962 (Petri, 1962). In a relatively short period of time, Petri nets were used extensively in practice as well as having seen a continuing theoretic development.

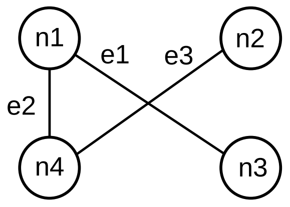

Figure 5: Simple Graph

A Petri net is a graph. That is, it is a set of nodes, edges and rules associating edges and nodes. Formally, a graph can be defined as a triple

$$G = (N, E, \varphi)$$

consisting of a set N of nodes, a set E of edges and a mapping of the elements of E to a pair of elements in N. Figure 5 shows a simple graph where

$$N = \{n1, n2, n3, n4\},\ E = \{e1, e2, e3\}$$

and a mapping function φ

$$\begin{gathered} e1 \rightarrow (n1, n3), \\ e2 \rightarrow (n1, n4), \\ e3 \rightarrow (n2, n4). \end{gathered}$$

A Petri net has two types of nodes: places and transitions. Places are graphically represented by ellipses and transitions by rectangles. Petri net edges are referred to as arcs and are always directed (i.e., they have a head and tail and are drawn as an arrow). Formally, a Petri net is a 4–tuple

$$PN = \{P, T, I, O\},$$

where P is the set of places, T is the set of transitions, $I(p, t)$ is a mappings of $P \times T$ and $O(t, p)$ is the a mappings of $T \times P$.

Colored Petri nets with time provide a complete language for the design, specification, simulation, validation and implementation of large software systems (Jensen, 1997a,b,c,d). Colored Petri nets combine the strength of Petri nets with the strength of programming languages. Petri nets provide primitives for the description of the synchronization of concurrent processes, while programming languages provide primitives for the definition of data types and the manipulation of data values (Jensen, 1997a).

It is, in particular, well suited for systems in which communication, synchronization and resource sharing are important. Typical application areas include communication protocols and distributed systems. Petri nets with time offer desirable qualities for characterizing and modeling the domain of distributed virtual simulation systems.

A SYSTEM MODEL

A Petri net simulator has been constructed to capture the temporal characteristics of tokens as they are removed and created by transitions in a system design. "Colored" tokens or tokens with an associated datatype encode time information into the token itself. For example, when a token is created, a "creation time" attribute is set. When a token is created in a place, a second time-stamp attribute called "arrival time" is set.

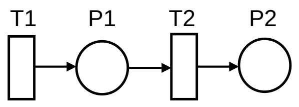

Figure 6: Petri Net Example

Consider the Petri net shown in Figure 6. In this network, tokens are automatically created by transition T1 because it has no input places, therefore, it is always "enabled". When a token is first created in P1, both the creation and arrival time attributes are set to the current system time (simulator time). After the "firing" of transition T1, T2 is enabled because a token now exists in its input place P1.

Transitions have the following rules. If a transition does not have any input place(s), it is assumed to be a generator or source of tokens. It will set the creation and arrival time of tokens generated at its output place(s). If it has an input place(s), the creation time is copied and left unaltered, but the arrival time will be updated to reflect when the token is placed into the output place(s).

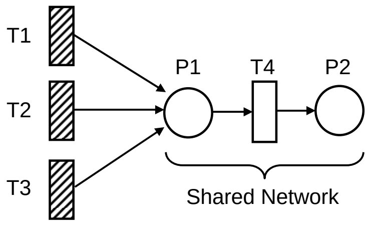

Figure 7: Model With 3 Logical Processes

The time associated with a transition is the time between when the transition is enabled and when the transition is fireable. A transition becomes fireable if and only if it is continuously enabled (i.e., the tokens in incoming places reside until the transition's delay is spent). The time delay can be specified by stochastic distributions. It can also be set to zero, meaning it will fire immediately when enabled. When multiple transitions are enabled and ready to fire, one is randomly selected. This random selection captures the asynchronous behavior of a system model.

Figure 7 shows a system model for three logical processes connected by a simple network. Each logical process is modeled by a transition (T1, T2 and T3) that serves as a source of message traffic (tokens). As tokens are generated and created in place P1, transition T4 (the network model) removes them and creates messages in place P2. The network in this case is modeled as a mover of messages with a fixed deterministic time. This time represents the total latency associated with moving the message from sender to receiver. Each message is assumed to be the same size, and the latency includes any sender overhead to transmit the message, the time of flight (propagation time), the transmission time (message size divided by bandwidth), and receiver overhead. With this simple model, the fundamental temporal characteristics of the data being exchanged of a distributed virtual simulation can be evaluated as it captures the asynchronous behavior of the logical processes and the non-deterministic behavior of the MAC (Media Access Control) layers of an Ethernet-based network.

SYSTEM SCALING

To describe or specify a system more compactly in terms of scalability, the concept of shared transitions and places are introduced along with a corresponding "folding" factor. A shared place represents a common resource. A shared transition can represent any global action. The folding factor defines how many times the unshared places and transitions should be duplicated (i.e., unfolded) to produce an executable

Petri net model. Shared places and transitions allows for more convenient system specifications to be described, and also facilitates the automatic creation of executable models for evaluation.

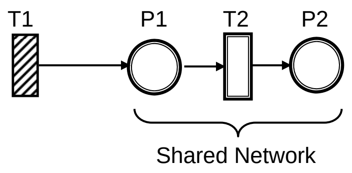

Figure 8: Model with Shared Resources

Figure 8 with three folds is identical to Figure 7, and captures the logical processes (simulators) and shared network infrastructure. The folding factor is useful as it represents a scaling of the system in terms of the number of interconnected simulators.

ENTITY STATE PDU GENERATION

To estimate the performance and scalability of DIS oriented system designs, a characterization of the message traffic generated by logical processes needs to be defined. The characterization used herein is based upon information provided by Durbach and Fourneau's research (Durbach and Fourneau, 1998). Their research investigated the DIS dead reckoning algorithms and collected entity state PDU data for an aircraft flying a "winding" route. A histogram of the interarrival patterns for this single entity is shown in Figure 9: 86% of the PDUs are transmitted, on average, every 0.2 seconds, 10% are transmitted every 0.1 seconds, and 4% are transmitted every 4.4 seconds.

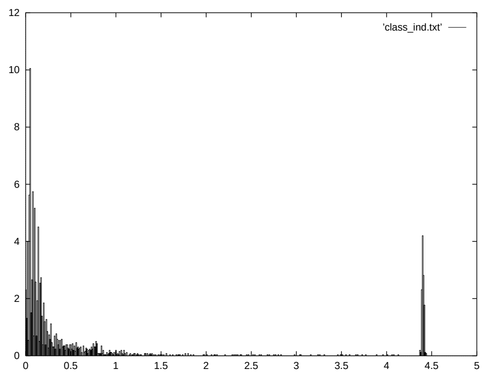

Figure 9: PDU Interarrival Histogram (Durbach and Fourneau, 1998)

A simple model to characterize this distribution can be specified as the time function associated with a Petri net transition. For example, it is used by transition T1 in Figure 8 to represent a simulator connected to a simple network infrastructure. Because transition T1 does not have any input places, this distribution effectively models the simulator as a source creating new tokens (entity state PDU messages).

ESTIMATING DIS PERFORMANCE

An initial estimate of DIS performance uses the model shown in Figure 8 with transaction T1 set to the entity state generation model. The performance of this system is in terms of two factors, scalability (folding) and network latency. Of primary interest to quantify performance is the temporal characteristics of the data (aging) arriving in place P2. Data arriving at place P2 effectively represents data received by each simulator.

The Mersenne Twister algorithm was used as a source for pseudo-random number generation. Each simulation run consisted of the processing of 500,000 events, which yielded approximately 250,000 messages passed between logical processes for each (scale, latency) data point. This equates to approximately 2.6 hours of simulated time for a system scaled for 10 simulators. Confidence intervals for the average age of data is very narrow due to the high number of tokens passed. Long runs minimized the effect of initial transients.

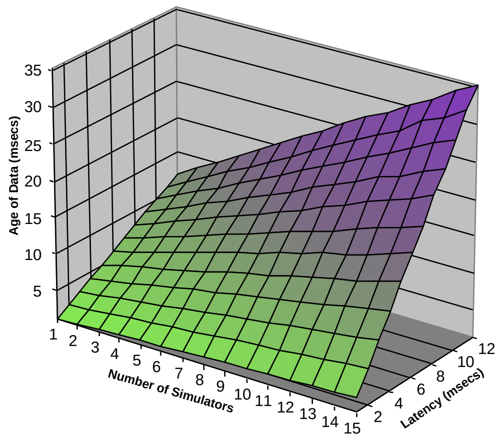

Figure 10: Aging Due to System Scale and Latency

Figure 10 shows the aging characteristics of the data within a stable operating region of the system considering scale and latency. As this graph illustrates, when 15 simulators are connected together with a network that takes 12 milliseconds to transmit a token, the average age of the data seen by each simulator will be about 35 ms old.

DIS EXAMPLE

As a practical application of this research, consider the evaluation of a DIS oriented system design where the latency of the network is 35 ms. This latency value is representative of typical networks used in distributed virtual simulation exercises.

For this example, we are only concerned about loosely coupled interactions as defined in the DIS standard (IEEE, 1995). As defined, the transport-to-transport latency value of 300 ms is of primary interest. Also of interest is the DIS acceptable reliability statistic of 95%. DIS defines the transport-to-transport latency and acceptable reliability as follows:

- Transport-to-transport latency: Total acceptable latency between any two simulators, from the input of the transport layer at the sending simulator to the output of the transport layer at the receiving simulator.
- Transport-to-transport acceptable reliability: Fraction of PDUs that must be delivered within the transport-to-transport acceptable latency, with remaining PDUs randomly distributed over the entire stream of PDUs.

As modeled in Figure 8, network transmission is the transport-to-transport latency which includes time consumed by all media, bridges, routers, gateways, encryption/decryption devices and intervening networks.

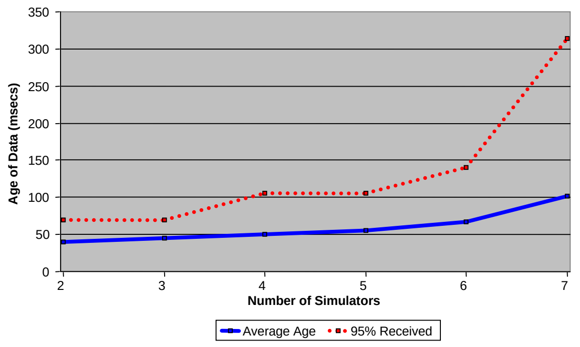

Figure 11: Age of Data for 35 ms Latency

Executing the simulation with a fixed 35 ms network latency and varying the scale from 2 to 7 interconnected simulators yields the plot shown in Figure 11. Connecting 6 simulators together results in each simulator receiving, on average, data that is about 65 ms old. Each simulator also receives 95% of the data exchanged within 140 ms. This simple analysis indicates the system can be scaled to include 6 simulators without violating the DIS standard for loosely coupled interactions.

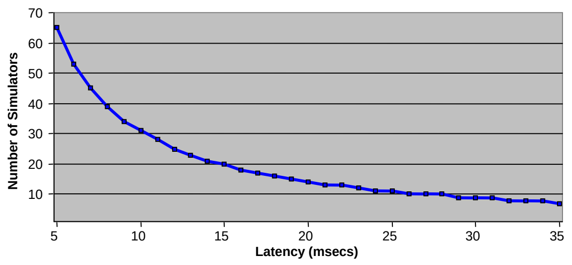

Figure 12: Scalability vs Latency for 95% Reliability

The results shown in the previous section indicates that scaling a DIS-oriented system beyond 6 simulators presents a problem for a network that exhibits a 35 ms delay in transporting entity state data. What if the network latency could be reduced? As shown in Figure 12, network latency plays a large role in the overall scalability of a DIS simulation. This figure shows the number of simulators that can be interconnected successfully and still meet the DIS standard for loosely coupled interactions. For example, if the network latency can be reduced to 5 ms, then a simulation that consists of 65 simulators that generate DIS entity state PDUs (as modeled) can be connected.

CONCLUSIONS

Zhou (Yi Zhou and DeFanti, 2000; Zhou et al., 1999) noted that standard practice for the design of distributed virtual simulations is basically trial and error, empirical, and totally lacks a formal foundation. This research addresses this issue by considering the performance and scalability of distributed virtual simulations in the context of the temporal requirements of the models used within the simulation. It has shown how Colored Petri nets with time can be used as a tool to evaluate the temporal characteristics of system designs. Even through a DIS oriented example was evaluated, the application of the research extends beyond DIS.

A DIS oriented example was modeled because it is a well established, popular standard in which to evaluate system scalability and performance characteristics. It also specifies the temporal requirements associated with conducting distributed virtual simulation studies. The standard defines the latency requirements of entity state PDUs to satisfy dead reckoning algorithms which in turn are designed to meet the requirements associated with training people. The results of this research indicates an upper bound for the number of simulators that can be interconnected and still meet the temporal requirements as specified by the standard for a network with a 35 ms latency. It also shows how scalability is related to network latency.

ACKNOWLEDGEMENTS

The authors would like to express gratitude to a few individuals at the SIMAF facility located at Wright Patterson AFB. Specifically, Randyll L. Levine and Christopher Buell for their thoughtful comments and inputs associated with this paper.

REFERENCES

Durbach, C. and Fourneau, J.-M. (1998). Performance evaluation of a dead reckoning mechanism. In *DIS-RT '98: Proceedings of the Second International Workshop on Distributed Interactive Simulation and Real-Time Applications*, page 23, Washington, DC, USA. IEEE Computer Society.

Fujimoto, R. M. (2000). *Parallel and Distributed Simulation Systems.* John Wiley & Sons, Inc.

IEEE (1995). *Standard for Distributed Interactive Simulation - Communication Services and Profiles (Standard 1278.2).*

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley and Sons, Inc.

Jensen, K. (1997a). A brief introduction to coloured petri nets. In *TACAS '97: Proceedings of the Third International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 203–208, London, UK. Springer-Verlag.

Jensen, K. (1997b). *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol. 1 Second Edition, Basic Concepts.* Springer-Verlag.

Jensen, K. (1997c). *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods.* Springer-Verlag.

Jensen, K. (1997d). *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol. 3, Practical Use.* Springer-Verlag.

Law, A. M. and Kelton, W. D. (2000). *Simulation Modeling and Analysis, Third Edition.* McGraw Hill.

Liu, J. W. S. (2000). *Real-Time Systems.* Prentice Hall.

Mosberger, D. (1993). Memory consistency models - tech report that is an update of paper published in ACM SIGOPS. *Operating Systems Review*, 27.

Peterson, J. L. (1977). Petri nets. *ACM Computing Surveys*, 9(3):223–252.

Petri, C. A. (1962). *Kommunikation mit Automaten.* PhD thesis, University of Bonn, Fed. Rep of Germany.

Singhal, S. and Zyda, M. (1999). *Networked Virtual Environments, Design and Implementation.* Addison Wesley.

Song, X. C. and Liu, J. W. S. (1995). Maintaining temporal consistency: Pessimistic vs. optimistic concurrency control. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):786–796.

Tanenbaum, A. S. (1995). *Distributed Operating Systems.* Prentice Hall.

Yi Zhou, T. M. and DeFanti, T. A. (2000). Modeling and performance analysis using extended fuzzy-timing petri nets for networked virtual environments. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, Vol. 30, No. 5, Pages 737-756.*

Zhou, Y., Murata, T., and DeFanti, T. (1999). Modeling and analysis of collaborative virtual environments by extended fuzzy-timing petri nets. *The Institute of Electronics, Information and Communication Engineers (IEICE) Transactions on Fundamentals of Electronics, Communications and Computer Sciences.*