A Distributed Scene Graph Approach to Scaled Simulation-Based Training Applications

Douglas B. Maxwell

U.S. Army Research Laboratory, Human Research and Engineering Directorate,
Orlando, FL
douglas.b.maxwell@us.army.mil

Joe Geil, William A. Rivera

University of Central Florida
Orlando, FL
jpg9, 14wrivera@knights.ucf.edu

Dr. Huaiyu Liu

Intel Research
Hillsboro, OR
Huaiyu.liu@intel.com

ABSTRACT

Current infantry training simulators are based on first person shooter gaming products and have been used for many years for individual and small unit training. There is a need for a broader application of simulation-based training systems to train multiple small teams in concert or larger unit operations. Additionally, the systems will need to accurately present the operational area with larger numbers of civilian and opposing forces. This requires a simulation-based trainer to scale from currently tens of users to hundreds of users and entities in the same virtual space at the same time. The biggest limiting factor for this activity has been the inability for the backend simulation architectures of the first person shooters to simultaneously broker the large numbers of entities needed to support the scaled simulation. The U.S. Army Research Laboratory's Simulation and Training Technology Center (ARL STTC) and the Intel Corporation entered into a Cooperative Research and Development Agreement (CRADA) in February of 2013 to address core simulation scaling issues. The ARL/STTC and Intel Corp. performed a series of five joint scalability experiments over the summer of 2013 to test new prototype architectures that support scaled operations. These scalability experiments were open to the public and included volunteers from industry and academia. The experiments were able to show significant increases in the number of humans who could log into a coherent training simulation and interact with each other while performing a mission. This paper will present the results of one of the events, including the data collected from the distributed simulators which were located at various locations across the continental United States. We will discuss the architecture of the prototype simulator, provide performance findings, the statistical approaches used to analyze this data and provide an interpretation of findings. Finally, we discuss a model developed from the autonomous agent simulator loads and compare it to the performance of the simulators when loaded with large numbers of human users.

ABOUT THE AUTHORS

Douglas Maxwell conducts research into the use of virtual environments for strategic applications as a Science and Technology Manager for the U.S. Army Research Laboratory, Human Research and Engineering Directorate, Simulation and Training Technology Center in Orlando, Florida. He is the director of the Military Open Simulator Enterprise Strategy (MOSES).

Joe Geil is PhD student in Modeling and Simulation at the University of Central Florida. He is also a Graduate Research Assistant at the STOKES Advanced Research Computing Center at UCF and a Mathematics professor at Valencia College.

William Rivera is a PhD student in the Modeling and Simulation program at the University of Central Florida. His research interests include predictive modeling, machine learning, software architecture and engineering and data mining with emphasis on implementing artificial intelligence into software applications.

Huaiyu Liu is a research scientist in the Security and Privacy Research group, Intel Labs. She holds a Ph.D. in Computer Sciences from the University of Texas at Austin. While at Intel, she had worked on multi-radio networks, energy efficient communications, and scalable infrastructure for multi-user virtual environments.

A Distributed Scene Graph Approach to Scaled Simulation-Based Training Applications

Douglas B. Maxwell
U.S. Army Research Laboratory, Human Research and Engineering Directorate,
Orlando, FL
douglas.b.maxwell@us.army.mil

Joe Geil, William A. Rivera,
University of Central Florida
Orlando, FL
jpg9, 14wrivera@knights.ucf.edu

Dr. Huaiyu Liu
Intel Research
Hillsboro, OR
Huaiyu.liu@intel.com

INTRODUCTION

Traditional game based military simulators use a monolithic approach to deploying training material. This means that a game level is provided on a single server where only a limited number of users are able to log in to each level. This vertical scaling prevents the ability to conduct larger scaled operations such as multiple teams training in concert or a company level engagement. Further, there is also generally no overhead left to provide for neutral participants or opposing forces.

This paper analyzes data collected from a joint US Army Research Lab / Intel Corporation experiment conducted on 21 Jun 2013. The data comes from a prototype simulation-based trainer that is currently distributed geographically across the United States. Participants in the experiment included industry and academic partners who were geographically dispersed evenly across the continental United States. The data illustrates the performance of the simulator and provide a means to construct a predictive algorithm for future deployments. High level questions this work will answer are: How many trainees could this architecture potentially support in ideal conditions? What are the correlated costs of resources used? Can those costs be predicted for future experiments?

Data collected during the exercise included network and server performance as well as memory and processor resource allocation. A large amount of data was collected, however not all of it was needed for this analysis. For the purposes of this project, the analysis is limited to the system demands on the major constituent pieces of the distributed simulator. Certain behaviors and trends are examined, which include the Central Processing Unit (CPU) and network bandwidth loads when subjected to the theoretical limits of the maximum number of users this system can support.

RATIONALE FOR A SCALED SIMULATION-BASED TRAINER

The Army Learning Concept 2015 (Morton, Lucious Department of the Army, HQ Deputy Chief of Staff, 2011) identifies the Army's need to develop adaptive, thinking Soldiers and leaders capable of meeting the challenges of operational adaptability in an era of persistent conflict. In response, ARL/HRED/STTC has identified a need for scalability and flexibility for next generation training applications, which need to handle more entities to create realistic scenarios in new operational environments.

Virtual environments show promise in providing more operationally accurate and persistent worlds for the soldiers to train in. To make this a reality, a virtual operational environment needs to support more than just the soldiers who are training, but also a faithful simulation of the opposing forces and neutral forces. This usually requires more than 100 users to operate in the same space at the same time to achieve a realistic mission. However, the majority of current simulation-based virtual training applications are only used to train at the small unit level, 40 soldiers or less. Hence, properly representing the operational environments for Army training needs requires a substantial increase in the scalability and flexibility of virtual environments.

Virtual Training Applications for Army Training Need

As identified by the Army Learning Concepts 2015 (Army, 2015), the main theme, *operational adaptability*, depends fundamentally on educating and developing leaders capable of understanding the situation and adapting actions. The Army must meet the challenge to prepare soldiers and leaders who are technically and tactically proficient, can think critically, make sound decisions, interact across cultures, and adapt quickly to rapidly evolving situations in full-spectrum operations. 3D Immersive Learning Environments, or Virtual World Simulations, provide a way to simulate complex problems and engage participants to learn the skills required to solve the problems. These simulations allow us to practice improvisational discovery (Burgos et al., 2007) and build an understanding of complex processes in order to find solutions.

Research suggests that we learn better and retain information longer when we are engaged in the learning (Yee, 2006). Virtual world simulations engage the player, especially when they are designed to facilitate unpredictable complex dynamics in virtual environments to help learners understand and succeed in complex systems. Operational accuracy can be defined by three overarching requirements: realistic numbers of participants given a certain scenario, realistic operational areas, and realistic complexity. Large virtual areas, high numbers of objects, and the complexity of their behaviors and appearance is key to virtual operational environment accuracy. As a result, scalability of virtual environment operations is essential to satisfy operational accuracy requirement.

The Military Open Simulator Enterprise Strategy Distributed Scene Graph

The joint US Army Research Lab/Intel Corporation team needed a distributed server architecture which was scalable, reliable, provided load balancing, fostered a persistent environment, had seamless server deployment and was cost effective. The available architectures that currently exist each had different advantages and disadvantages, none of which fully addressed all of the team's requirements. These architectures are briefly described below. To this end, the team leveraged Intel's Distributed Scene Graph (DSG) technology utilizing open source code that turned out to be a significant benefit because this code was open source, which rendered the activity cost effective.

U.S. Army's Simulation and Technology Training Center (STTC) and Intel Labs collaborated to test an experimental distributed simulation-based training environment using large numbers of human users to properly stress the system. The environment is a blending of the MOSES (Military Open Simulator Enterprise Strategy) environment and an integration of Distributed Scene Graph (DSG) software. Two experiments were devised to sample the distributed system load under various usage scenarios and compare those samples to synthetic benchmarks. An example of the synthetic benchmarks used were the instantiation of hundreds of objects in the scene, then assigning physical attributes to the objects, then allowing the objects to interact. These interactions were then used as load stressors for the physics engine.

DISTRIBUTED SCENE GRAPH ARCHITECTURE

The MOSES Grid is derived from OpenSim simulators (OpenSim, 2014). OpenSim, like other existing virtual worlds and multi-user online games, adopts a traditional architecture as shown in Figure 1. The core of this architecture is a set of simulators that each hosts a virtual geographic area of responsibility. In each simulator, a set of heterogeneous simulation engines (for instance, physics engine and script engine) are tightly integrated in the same simulator process. This traditional architecture inherently limits the scalability of operations on the same virtual land to the capacity of a single server. As demonstrated in (Liu et al., 2010),when the number of interacting entities increases on a virtual land, the computation and communication load increases quadratically and can quickly exceed a single server's capacity and deteriorate the quality of real-time simulation and visualization dramatically. In addition, when a simulator crashes, all operations on the virtual land are down and the land becomes inaccessible to users. Another limitation of the "all-in-a-box" architecture is the mismatch between the homogeneous architecture of simulators and the heterogeneous characteristics of simulation engines (Liu et al., 2010). It does not have the flexibility to map the heterogeneous engines to different hardware that fit their different compute or communication characteristics.

Distributed Scene Graph Architecture Design

We present the Distributed Scene Graph (DSG) in Figure 1, to address the scalability limitation of traditional architecture. In general, we view a virtual world as a collection of the "*Scene*" and "*the actors*" operating on the Scene.

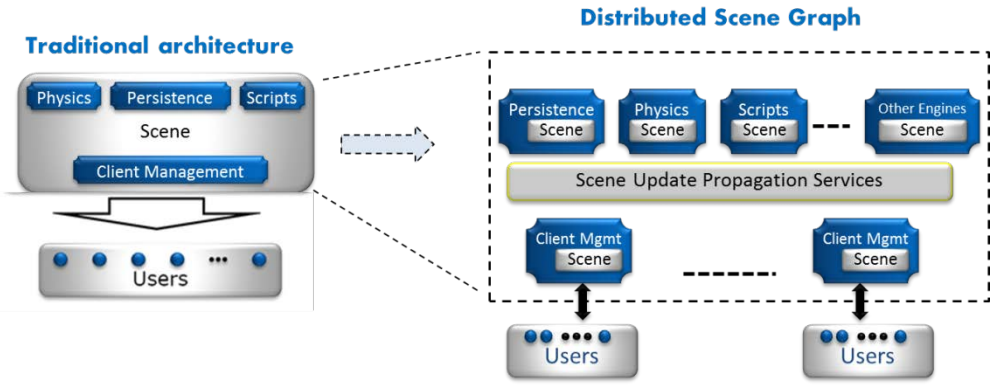

Figure 1. Traditional Monolithic versus DSG Architecture

The Scene is the data structure that stores and manages the state of a virtual world, including the complete set of all entities (objects and avatars) and their properties. When the value of any property is changed, we say that the Scene is updated or an update is generated on the Scene. The Actors are a set of simulation engines operating on the Scene, where we refer to each simulation engine as an actor. Examples of actors include: Physics Engine Actor (PEA) that simulates object behaviors following physical rules, Script Engine Actor (SEA) that executes scripts to drive object behaviors, Client Management Actor (CM) that update the Scene as indicated by user inputs and send updates to users to refresh their views, Persistence State Actor (PSA) that persists the Scene (e.g. periodically storing Scene to database), and maybe also some special purpose simulation engines such as cloth draping simulation.

Based on the above generalized view of a virtual world, the key idea in DSG is to decouple the Scene and the actors operating on the Scene, where actors operating on the same Scene can run on different servers with configurations matching the need of the actors. For performance reasons, each simulator maintains a local copy of the Scene. To maintain a consistent view of the Scene across simulators, DSG introduces an update propagation service on every simulator to propagate updates and events to synchronize their view of the Scene. To provide concurrency control, a time-stamp based approach is introduced to resolve update conflicts, with the assumption that the clocks of all simulators are synchronized with certain accuracy. For example, NTPv4 can usually maintain clock synchronization at the level of 10 milliseconds over the long distance WANs (NTP, 2008).

The DSG architecture itself does not assume any particular topology on how the simulators are connected for implementing the update propagation service. In practice, the simulators can be connected into a tree topology and the DSG design supports the tree topology. A major benefit of the DSG architecture is that it enables each actor operating on the Scene to be independently scaled and load balanced by adding new hardware dynamically. For example, for client managers whose operations are communication intensive, DSG makes it possible for them to be detached and run on hardware that is provisioned for high speed networking. In fact, we demonstrated that by spreading clients across multiple client managers, a DSG system was able to support 1000 users to interact concurrently (Lake et al., 2010).

Experiment Rationale

The first experiment scenario was designed and executed in the MOSES virtual environment in a simulation called Atropia, which contains culturally representative content and role players from a fictional area near the Caspian Sea and is designed for learning of cultural knowledge through role playing. Intel Labs' DSG technology, a scalable virtual environment architecture, is used in ARL's MOSES grid to support more than 100 global users participating in the same virtual training session. In general, the MOSES DSG Experiments are designed to achieve proof of

concept that more than 100 – and in future experiments more than 1,000 – users can operate in the same virtual environment at the same time while participating in realistic training missions. Additionally, the data collected using human volunteers was compared to baseline tests of similar scale using non-player characters.

The second experiment was a baseline event executed in March of 2014 that involved generating 60 non-player character (bot) agents on a non-DSG enabled MOSES server. The purpose of this event was to generate performance data on the simulator CPU and bandwidth to compare to the observations made under similar circumstances with human controlled bots. There were four data collection activities. The first activity involved generating 60 bots on a simulator with flat terrain and no content (3D models or textures) with no time allotted for simulator settle. The bots were all logged in, then shut down. The second activity involved generating 60 bots on a simulator with flat terrain and not content, but allowing for 10 minutes of settle time. The third activity involved generating 60 bots on the Atropia terrain with a complete replication of the human scene. This scene was a copy of the scene graph used in the June 21, 2013 event. The purpose of using this scene graph was to compare the agent's load on a simulator with the same content and terrain as the human experiment. The last baseline experiment was to generate 60 bots in the Atropia scene and add 200 physical objects to the scene. The objects were spheres that were programmed to have physical properties and collide with each other. This purpose of the addition of the spheres was to exercise the physics engine actor and generate more load on the simulator.

Method of Data Collection

The MOSES Distributed Scene Graph data discussed in this section was collected in June of 2013. Processor loads for all the actors and network bandwidth consumed between the client manager actors and the persistence state actor was sampled at regular intervals and written to log files. In order to analyze the MOSES data, it was necessary to prepare the data in such a way as to address the large amounts of missing data from some of the servers. Since client manager one (CM1), the persistence state actor (PSA), script engine actor (SEA) and physics engine actor (PEA) were all located on site, the data was complete. But the data for the remote servers CM2, CM3, and CM4 was missing values sporadically throughout the excel spreadsheet. To address this, it was decided that the data would be reduced to 1 minute increments instead of the original 10 second increments, which would allow for the creation of a complete data set for all servers but CM2.

In reducing the time to 1 minute intervals, every six samples were condensed down to a single averaged sample in the following fashion. For the numbers of agents who were currently logged in, the mode was used to represent the number of agents within that minute. This was ideal since the number of agents does not vary all that much within a single minute. In the few instances where the mode could not be determined due to having two or more modes, the median was used to decide the number of agents for that particular minute. The median was chosen over the mean since it tends to be less affected by outliers and data that is skewed.

To determine the CPU load for the minute increments, it was decided that the CPU load percentage occurring closest to the actual minute recorded in the file would be used. These times occur at exactly 3 seconds after each new minute. For the bytes collection data, the values are summed up as they go down the columns. So for each minute's throughput, the difference was taken between the largest and smallest bytes tallies within the 60 second range, in order to get the number of actual bytes within that minute. This allowed for the total bytes received and sent to be calculated even when one of the cells was missing data. Once these two columns were completed, another data set was created by summing the bytes received and the bytes sent for each minute, in order to calculate the total bytes for each minute.

To address the comment made earlier in regards to the CM2 data set, there was a time interval of seven minutes in which no data on the number of agents was collected. Also, within three of those seven minutes, the CPU load, bytes received, and bytes sent were all reported to be at zero for the entire seven minutes. This was due to the client manager crashing during the simulation. It was able to reboot and allow agents back into the simulation within a relatively short time span, but because of this downtime, the minutes from 10:45 PM until 10:52 PM have been removed from the CM2 data set. But for graphing purposes within MATLAB, the missing minutes needed to be included so that all servers had the same number of minutes. Thus, there is a noticeable drop in the graph down to zero during this aforementioned time span for client manager 2.

MOSES DSG Performance Correlation

It was necessary to determine if the correlations between our variable were linear or not. The variables examined were number of agents, CPU load, network bytes received, network bytes sent, and total network bytes transmitted. When comparing two variables in the correlation coefficients matrix, ratio values close to 1 represent a strong linear correlation, whereas values close to zero represent no linear correlation. Coefficients above 0.5 will be considered as possible candidates for linear correlation, although further analysis would be necessary to determine how strong a linear relation actually exists between the two variables. The value 0.5 was chosen due to the results of the calculated p-values, which determined that values greater than or equal to 0.5 were statistically significant for linear correlation. The MATLAB output for the correlation coefficient matrix can be found in Table 1.

Table 1. Server Correlation Coefficient Matrices

CM1	**Agents**	**CPU Load**	**Bytes Received**	**Bytes Sent**	**Total Bytes**
Agents	1	0.835923	0.623667	0.04537	0.600043
CPU Load	0.835923	1	0.764727	0.241768	0.775158
Bytes Received	0.623667	0.764727	1	0.147002	0.977838
Bytes Sent	0.04537	0.241768	0.147002	1	0.350832
Total Bytes	0.600043	0.775158	0.977838	0.350832	1
PEA	**Agents**	**CPU Load**	**Bytes Received**	**Bytes Sent**	**Total Bytes**
Agents	1	0.167292	0.47897	0.132378	0.47522
CPU Load	0.167292	1	0.006874	0.021217	0.008707
Bytes Received	0.47897	0.006874	1	0.312337	0.995683
Bytes Sent	0.132378	0.021217	0.312337	1	0.399167
Total Bytes	0.47522	0.008707	0.995683	0.399167	1
PSA	**Agents**	**CPU Load**	**Bytes Received**	**Bytes Sent**	**Total Bytes**
Agents	1	0.479049	0.58472	0.077115	0.584772
CPU Load	0.479049	1	0.687381	0.125331	0.689674
Bytes Received	0.58472	0.687381	1	0.098605	0.997947
Bytes Sent	0.077115	0.125331	0.098605	1	0.162143
Total Bytes	0.584772	0.689674	0.997947	0.162143	1
SEA	**Agents**	**CPU Load**	**Bytes Received**	**Bytes Sent**	**Total Bytes**
Agents	1	0.494382	0.510509	0.152269	0.50706
CPU Load	0.494382	1	0.611977	0.247083	0.613944
Bytes Received	0.510509	0.611977	1	0.327457	0.996003
Bytes Sent	0.152269	0.247083	0.327457	1	0.410541
Total Bytes	0.50706	0.613944	0.996003	0.410541	1

For client manager 1, there is evidence of a linear correlation between the number of agents and the CPU load, between the number of agents and the network bytes received, and between the number of agents and the total network bytes. Also, there is evidence of a linear correlation between the CPU load and the network bytes received, and between the CPU load and the total network bytes. Client managers 2, 3 and 4 had identical hardware set ups, resulting in results that were similar to client manager 1.

For the PEA, there are no correlation coefficients over 0.5 between the number of agents and any of the other four variables, although bytes received and total bytes are close. There is very little evidence of linear correlation between the CPU load and the other four variables.

For the PSA, there is evidence of a linear correlation between the number of agents and the network bytes received, and between the number of agents and the total network bytes. Also, there is evidence of a linear correlation between the CPU load and the network bytes received, and between the CPU load and the total network bytes. The

variables for number of agents and the CPU load have a coefficient that is close to 0.5, and therefore should warrant further examination.

For the SEA, there is evidence of a linear correlation between the number of agents and the network bytes received, and between the number of agents and the total network bytes. Also, there is evidence of a linear correlation between the CPU load and the network bytes received, and between the CPU load and the total network bytes. The variables for number of agents and CPU load have a coefficient that is very close to 0.5, and therefore should warrant further examination.

RESULTS

Figure 2 shows the data for all client managers. This data was used to graph the number of human agents within the system and the system load from the beginning until the end of the simulation run. For the number of human agents, the number tops out at 63 human agents present at one time within the simulation. One area of interest for this experiment was how the system would respond to having a large number of people logging into and interacting with the simulation. It can be seen that the trend lines (dashed lines in the graph) are all well below the 50% System Load mark, even when the number of agents is at its highest. The trend line for Client Manager 1 peaks the highest with a maximum value of about 36.3%. This would suggest that more agents could have been brought into the simulation, a promising observation as one of the key goals for the project is scalability. Also, when combining the trend lines for the three west coast servers, CM2, CM3, and CM4, they peak at a maximum value of about 35.6%. It turned out that these three servers combined have a trend line similar to that of CM1. This showed a balance of system load between the east coast and west coast servers. One peak of interest involves the Client Manager 2 CPU Load. At the 288th minute this server reaches 100.0% System Load. This is about seven minutes after the server came back online after crashing, there was a lot of logging back in by the participants occurring around this time, causing the spike in the CPU usage.

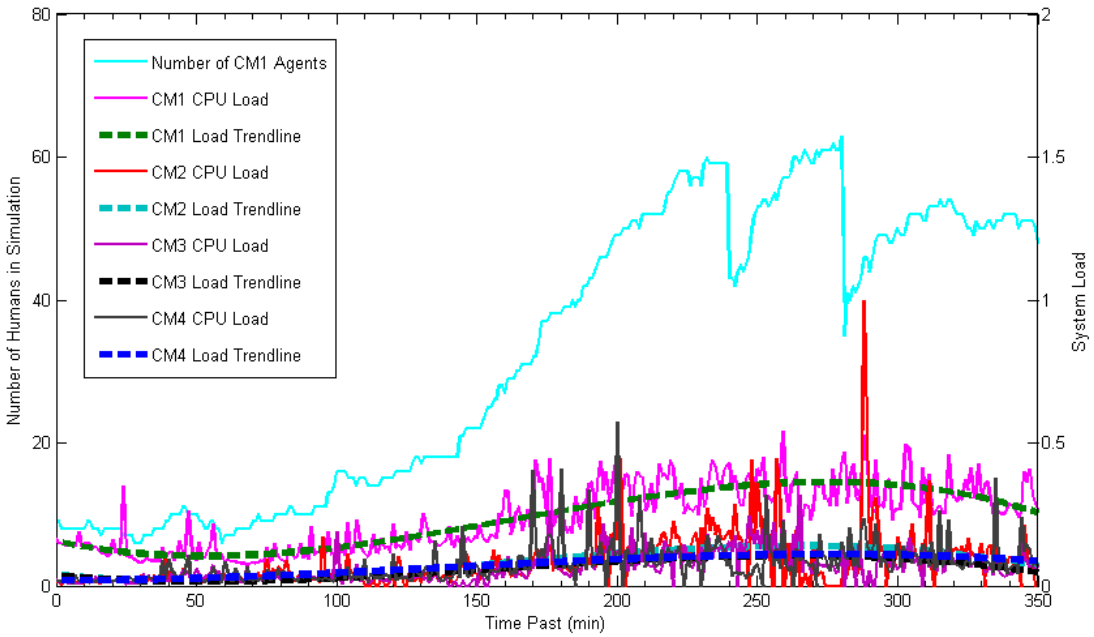

Figure 2. System Run Time vs. # Human Agents vs. System Load

Human vs Autonomous Bot Comparisons

One area of interest was the use of bots in the simulation to assist in the prediction of how the system would behave when actual human agents are introduced into the environment. Specifically, could the amount of CPU usage and the total network bandwidth for human agents be predicted using the numbers gathered by the virtual agents? The purpose of this is to reduce the amount of labor needed to test the system. Scatter plots were created to see if there were any similarities between humans and bots in regards to their CPU usage and their bandwidth usage. The scatter plots for the CPU load are shown in Figures 3 and 4.

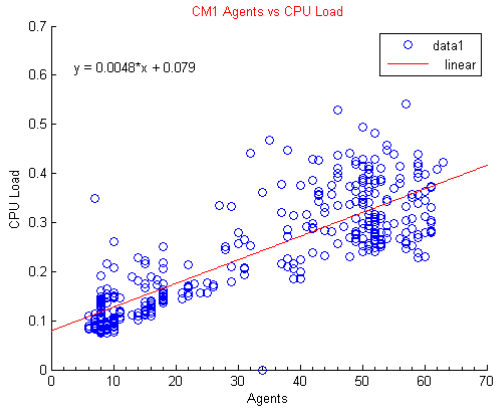

Figure 3. Agent's vs CPU Load

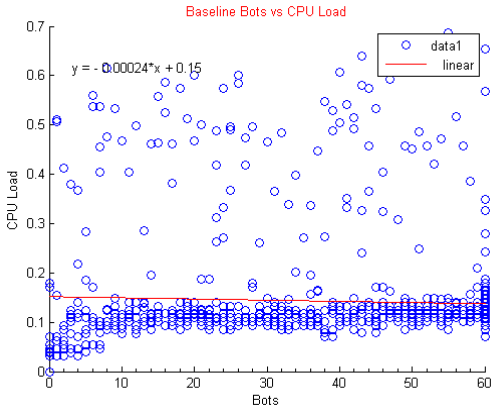

Figure 4. Baseline Bots vs. CPU Load

The live experiment (figure 3) shows an obvious trend in the data. As the number of agent's increases, the CPU load also increases in a near linear fashion. For the all bots baseline experiment (figure 4), there is not the same increasing linear trend line. The majority of the data form a near horizontal line, showing very little change in CPU usage as the number of bots was increased. There is also evidence of high CPU Load for small numbers of bots and low CPU Load for large numbers of bots. The same cannot be said about the live agents. From these results, it would seem that virtual bots are a poor choice in modeling the CPU usage behavior of human agents.

The scatter plots for the total bandwidth consumption are shown in Figures 5 and 6. In the live experiment (figure 5), there is an obvious trend in the data. As the number of agents increases, the bandwidth consumption also increases in a slight linear fashion. For the all bots baseline experiment (figure 6), there is not the same increasing linear trend line. The majority of the data form a near horizontal line, showing very little change in bandwidth as the number of bots was increased. The magnitude of the human bandwidth consumption is much smaller than the magnitude of the bots bandwidth consumption. Humans are not exceeding 12000 bytes, which is much smaller than the roughly 250,000 bytes that the majority of the bots' bandwidth lie. Once again, it would seem that the virtual bots' usage of the bandwidth is not representative of the behavior human agents are showing.

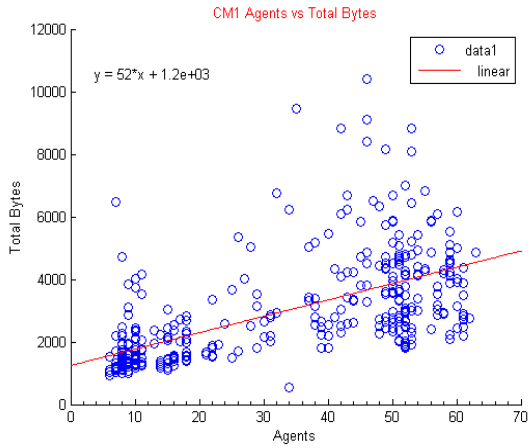

Figure 5. Total Bandwidth for Human Agents

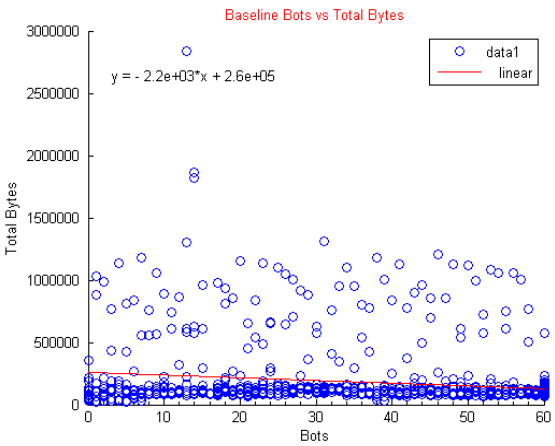

Figure 6. Total Bandwidth for Bots

Performance Prediction Model

One of the considerations regarding the data was the ability to predict the cost of bandwidth for future experiments. Exploring the linear relationship between time, agents and bandwidth encouraged the use of a linear regression model which provides simple good performance prediction models especially when dealing with small numbers of training cases (Friedman, Hastie and Tibshirani, 2001). The simple linear regression model is:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots \beta_n x_n + \varepsilon \quad (1)$$

Where $\hat{y}$ represents the expected value, β_0 represents the intercept, $\beta_{1\ldots n}$ represents the slope or coefficient for variable $x_{1\ldots n}$ and ε represents random error which is typically considered independent and identically distributed with a mean of 0. Accuracy may be determined by assessing the sum of square errors compared against the actual values. Deriving the coefficients (slope and intercept term) is provided via a closed form solution shown below in matrix notation for conciseness:

$$\beta = (X^T X)^{-1} X^T Y \quad (2)$$

Where $Y \ni \{\mathbb{R}^k\}$, $X \ni \{\mathbb{R}^{kxn}\}$ and $\beta \ni \{\mathbb{R}^n\}$. This formula holds if the column in the first position of the matrix has all its values set to 1 which allows for the intercept term to be included and solved for. The error term is excluded from the formula since it has an expected value of 0.

The baseline experiment conducted March 2014 used virtual agents in three different scenarios based on activity: Light, Moderate and Extreme. In both the baseline experiment and real world simulation bandwidth usage fluctuated over time with high amounts of variance. In real world simulations predicting bandwidth usage is also made difficult because there may be various levels of activity with numerous agents occurring simultaneously as well as network and hardware limitations.

To generate the final model linear regression was applied to the base case experiment for both the extreme and light scenarios and then the coefficients were averaged under the loose assumption that it may represent the average activity for a scenario.. During data analysis a high correlation between CPU as well as bandwidth led to the assumption that number of agents captures the complexity imposed by human agents although separating it is made difficult. Additionally the observed data indicated that time was a relevant variable to capture and may be indicative of human activity over time.

High correlation between time and number of agents suggested that both measures should not be used in the model and appropriate alternative included either removing one of the variables or combining them. In favor of keeping both they were combined and adjusted to account for some of the variance in the observed data from the human based experiments and also reduced the dimensions of the model. This resulted in the following linear regression model:

$$\sum_{i=1}^{n} f(x_i) = \beta_0 + \beta_1 x_i, \quad \{0 \leq x_i \}, \ \beta = \{1540, .196\} \quad (3)$$

Where x represents the elapsed time in minutes multiplied by the average number of human agents at that minute. Determining future complexity is difficult as it depends mostly on the physics and number of users present among other things. Including the user agent and time combination into the equation allows for the model to capture complexity based on the number of users at a specific point in time.

The model was then compared against each client managers (CM1, CM2, CM3 and CM4) data. Attempting to evaluate the model was difficult because the error rate will grow over time. If the predicted amount is off by a few bytes at each interval it will continue to increase and become unstable over time. For this reason the main evaluation criteria was based on how well the model compared against the linear regression model of the observed data. In a matter of speaking if the prediction performed at least as good or good enough as compared to the linear regression model then it may serve to predict the future linear regression models for future experiments. The difference

between the linear regression model for each client manager and the prediction model can be seen in table 2 and also in Figures 8 and 9 below.

Table 2. Linear Regression Errors

	CM1	CM2	CM3	CM4	Total
Model Error	1.73E+05	1.31E+06	7.33E+05	9.43E+05	3.16E+06
Regression Error	2.5E-10	7.55E+05	1.38E-09	1.09E-09	7.55E+05

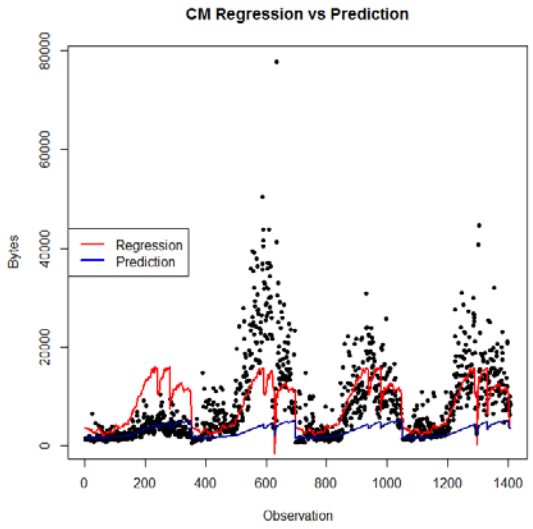

Figure 7. CM Regression vs. Prediction

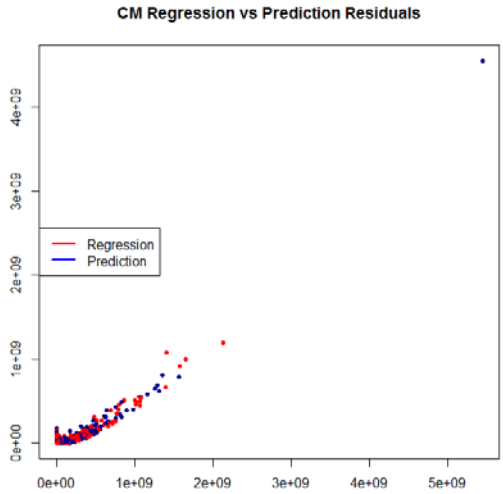

Figure 8. CM Regression vs. Prediction Residuals

It's important to mention that the model does under fit the data and had there not been an issue with CM2 than the difference in prediction accuracy would be considerable based on total error. One could argue that future real world experiments may encounter similar problems and therefore the model assumption is justifiable.

The ability to predict future bandwidth would be more feasible given more data from future experiments that included additional information such as what users were doing at specific time intervals. It may be possible to extract variables from further virtual experiment and then attempt to identify the same patterns in the human experiments as a way to build more reliable predictions.

CONCLUSIONS AND FUTURE WORK

The vision for future usage of virtual world technology in the infantry soldier training cycle is to create a simulation-based training system that can be used to exercise their classroom training and get a pass/fail grade from a trainer based on performance instead of a paper exam. The theory is we can create virtual environments with the richness and fidelity needed to properly exercise critical thinking skills and allow soldiers to exercise their classroom training. This research has shown it is possible to break the scalability barriers to allow for multiple small units to train in concert or for larger units to train for more complex missions.

This paper challenges the assumption that autonomous agents are the same as human agents when attempting to gauge simulator load and resource allocation. The system and network demands of human agents operating within a

simulation-based training system are vastly different as compared to artificial (bot or NPC) agents. Predicting future experiment network activity is made difficult because of human behavior and network and hardware limitations. During any experiment the amount of agents and activity may trend negatively or positively making it difficult to develop a model that generalizes well. We developed a model from an experiment using autonomous agents and compared it with bandwidth generated from human agent based simulation over time in a linear fashion. Refinement is required to increase accuracy of the model; however they are useful for determining future bandwidth needs as compared when performing linear regression. Additionally, we compared a distributed simulation server approach to a traditional "all in one" simulator. The distributed server shows promise that scaled simulations are possible in order to achieve a coherent virtual environment at high fidelities with complete interactions with all participants.

On the technology front, there are several areas to further improve the MOSES-DSG system load models. First, work is ongoing to improve the DSG implementation to provide robust performance. Second, work is ongoing to enhance the DSG design to support even larger scale of training sessions and no longer be constrained by the OpenSim region size. The next DSG architecture itself has no assumption of the size of a virtual environment. The team is developing an extension to DSG to enable fine-grained space partitioning, so that workload in a large-size virtual environment can be partitioned and mapped to different simulators for load balancing with fine-granularity. Another area for improvement is the login process. In future work, we plan to perform more human and artificial agent load tests to gather more performance data with even more nodes distributed to more geographically diverse sites.

ACKNOWLEDGEMENTS

The authors would like to express gratitude to the open source community, the volunteers who gave their time to participate in the technology evaluation events, and the MOSES participants who provided insight and assistance with the preparation of the virtual environments.

REFERENCES

Burgos, D., Tattersall, C., & Koper, R. (2007). Re-purposing existing generic games and simulations for e-learning. *Computers in Human Behavior*, *23*(6), 2656–2667. doi:10.1016/j.chb.2006.08.002

MathWorks (2014). *MATLAB. The Language of Technical Computing.* Retrieved April 12, 2014, from http://www.mathworks.com/products/matlab/

Morton, Lucious (Department of the Army, HQ Deputy Chief of Staff, G. (2011). *The U.S. Army Learning Concept for 2015* (pp. 1–72). Fort Monroe.

Network Time Protocol: Best Practices White Paper. 2008. Retrieved March 13th, 2014, from http://www.cisco.com/c/en/us/support/docs/availability/high-availability/19643-ntpm.html

Lake, D., Bowman, M., & Liu, H. (2010) Distributed Scene Graph to Enable Thousands of Interacting Users in a Virtual Environment. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games* (NetGames '10).

Liu, H., Bowman, M., Adams, R., Hurliman, J., & Lake, D. (2010) Scaling virtual worlds: Simulation requirements and challenges. In *Proceedings of Winter Simulation Conference*, 2010.

Morton, Lucious (Department of the Army, HQ Deputy Chief of Staff, G. (2011). *The U.S. Army Learning Concept for 2015* (pp. 1–72). Fort Monroe.

Network Time Protocol: Best Practices White Paper. 2008. Retrieved March 13th, 2014, from http://www.cisco.com/c/en/us/support/docs/availability/high-availability/19643-ntpm.html

OpenSim, (2014) http://opensimulator.org/wiki/Main_Page

Yee, N. (2006). The Demographics , Motivations , and Derived Experiences of Users of Massively Multi-User Online. *Presence: Teleoperators & Virtual Environments*, *15*(3), 309–330.S.R.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1). New York: Springer Series in Statistics.