Supporting Distributed Simulation on Scalable Parallel Processor Systems

Richard Williams
BMH Associates, Inc.
Norfolk, VA
Williams@bmh.com

John J. Tran
Information Sciences Institute, USC
Marina del Rey, CA
jtran@isi.edu

ABSTRACT

The Distributed Continuous Experimentation Environment (DCEE) is a permanent simulation infrastructure being set up by U.S. Joint Forces Command (JFCOM) to support Joint experimentation. DCEE will combine simulations running on both local JFCOM networks and Scalable Parallel Processor (SPP) networks. JFCOM has been working to develop tools to manage a large number of simulation computers with a minimal number of technical support personnel. These tools allow an operator to start and stop various applications, monitor and graph machine resources, generate simulation routing topologies, check network connectivity, and perform these functions in a secure environment.

As DCEE planning continues, the requirement for centralized federation control becomes obvious. The challenge of remotely coordinating the operation of hundreds or possibly thousands of simulation applications looms ever larger. The fact that numerous machines may exist on remote networks further complicates this issue. As an integral element of DCEE, centralized control will need to be expanded to manage and monitor SPP networks along with existing systems.

This paper will address the complex challenges of controlling and monitoring an extensive simulation environment. The paper will introduce the environment, describing the simulations and the SPP. The paper shall also discuss the operational and technical advantages using a centralized set of tools. The paper will not only examine the challenges encountered by attempting to run simulations on SPP networks, but also how these challenges are met.

ABOUT THE AUTHORS

Richard Williams is a Software Engineer with BMH Associates, Inc., supporting Joint Forces Command (JFCOM) in Suffolk, Virginia. He received a B.S. in Computer Science from the University of Central Florida. He has supported federation development for Attack Ops 00 (AO00), United Vision 01 (UV01), and Millennium Challenge 02 (MC02). He is currently supporting work for the Joint National Training Center (JNTC) and Distributed Continuous Experimentation Environment (DCEE).

John J. Tran is a Computational Scientist at the Information Sciences Institute, University of Southern California. He received both his BS and MS Degrees in Computer Science and Engineering from the University of Notre Dame, where he focused on Object-oriented software engineering, large-scale software system design and implementation, and high performance parallel and scientific computing. He has worked at the Stanford Linear Accelerator Center (SLAC), Safetopia, Inc., and Intel Corporation. While at SLAC, he was part of the E&M research team, whose research focuses on solving Maxwell's electromagnetic equations. While at Safetopia, he worked on a high performance encryption file-system, multi-processing communication abstraction layer, and distributed computing and dB interface. At Intel he was responsible for a software tool that performs noise analysis using Intel's circuit design methodology. His current research centers on Linux cluster engineering, effective control of parallel programs, and communications fabrics for large-scale computation.

Supporting Distributed Simulation on Scalable Parallel Processor Systems

Richard Williams
BMH Associates, Inc.
Norfolk, VA
williams@bmh.com

John J. Tran
Information Sciences Institute, USC
Marina del Rey, CA
jtran@isi.edu

INTRODUCTION

The Distributed Continuous Execution Environment (DCEE) is a permanent infrastructure being set up by the Joint Force Command (JFCOM) to support Joint Experimentation. The DCEE will merge distributed simulation systems running at JFCOM with simulations running on Scalable Parallel Processor (SPP) Networks. The reason for merging these systems is to add flexibility in the allocation of resources for use in very large scale simulations.

The government owns a number of SPP systems. These systems represent a very large resource that has yet to be regularly tapped into an interactive computing environment. The SPP systems typically have a number of strengths, such as high speed networks and high quality machines, which make them very desirable for the simulation community. However, they are sufficiently different from the typical simulation environment to present new and difficult challenges.

The Synthetic Forces (SF) Express (Burnett 1997) project recognized that increasing the size and complexity of distributed simulation greatly increases the importance of resource scheduling and allocation. The problem of preparing distributed systems to support simulation is simple to understand and tedious to resolve. Very large simulations can easily become impeded by details. Setup, configuration, installation and execution can easily become difficult tasks if a well designed system is not in place. This paper describes the system we have created to address these issues.

SPP Background and Motivations

The use of the Joint Semi-Automated Forces (JSAF) application on SPPs has its roots in the SF Express project (Burnett, *et al*, 1998) that achieved major milestones in terms of escalating simulation entities counts to an unprecedented level. At the same time SF Express laid down a foundational proof-of-concept that simulation experiments can be conducted over a wide area network (WAN). Operationally, the earlier SF Express experiments spanned across multiple SPPs and were semi-automated; in particular, the resources for each of the experiments were dedicated (reserved) for each scheduled simulation event.

A natural progression that builds on the earlier successes of the SF Express project includes: (1) further increasing the entity counts to match realistic synthetic operational theaters and urban environment, (2) further automate the process and management of simulation events, and (3) further optimize and enhance communication and dataflow by increasing flexibility and network connectivity abstractions.

The modernization of SPP resources, such as the increase in readily available bandwidth between SPP sites (DREN 2003) and the proliferation of freely available OpenSource operating system software (Linux) running on commodity hardware (Intel-based x86 architecture), have helped us increase high performance computing capabilities. The Joint Experimentation Scalable Parallel Processor (JESPP) team set new entity record counts, exceeding the million clutter entity count, with our experiments at the USC cluster in December 2002.

As previously mentioned, the increase in resources (especially those spanning multiple sites) necessitates that the event organizers automate the setup and simulation event as much as possible.

In the earlier implementations of JSAF applications, the communication layer is based on a broadcast model which for inter-SPP communications is not at all possible. Thus, design and implementation network topology abstraction serves as a means to further optimize intra-SPP and inter-SPP communication flow.

Participating Sites and Configuration

The SPP configuration for the Maui High Performance Computing Center (MHPCC) and Advanced Simulation Center (ASC) are both Linux clusters with dual processor P3's with one to two gigabytes of memory and approximately ten gigabytes of local scratch disk. Both operational sites at the Information Sciences Institute (ISI) and JFCOM run JSAF applications and our

controlling/monitoring tools on Linux workstations. The connection between ASC and MHPCC is a high-speed DREN network with sustaining bandwidth of 40 megabits/second (see Figure 1).

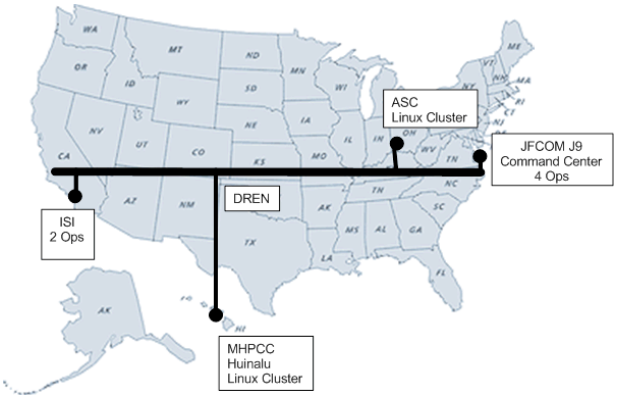

Figure 1 - SPP and operations centers

As noted, the multiple SPPs integrate to form a large simulation system with each SPP being a self-contained cluster of compute nodes. We strive to keep the framework of operation within each individual cluster consistent, despite varying policies amongst clusters.

The SPP Federation

The federation we are trying to support for the SPP testing is comprised of a number of federate applications bound together by a communication protocol called the Runtime Infrastructure (RTI) (Kuhl 1999).

The Federates

Currently there are a small number of applications being tested in the SPP configuration. These include JSAF, Clutter, and Simulation of the Location and Attack of Mobile Enemy Missiles (SLAMEM). These simulations are used to produce entity-level platforms which interact to produce a high quality synthetic environment.

JSAF is a high-fidelity entity level simulation, that can be used in a front-end/back-end configuration where the front-end (or GUI) will be on a local machine, while the back-end will be on the SPP. Entities can be instantiated on the front-end, by either an operator or from a file, but the actual simulation will be on the back-end within the confines of the SPP. This is done to limit the application-to-application communications within the confines of the SPP and to minimize Wide Area Network (WAN) traffic. JSAF also can be used in a "Pocket" configuration where both the instantiation and simulation of entities will be within the confines of a single local machine.

Clutter is a simulation used to add very large amounts of low-fidelity civilian and military traffic to a simulation to confuse sensors and create a more realistic simulation. Clutter will publish a large number of entities but only subscribe to a few interactions.

SLAMEM is used to simulate various operational sensors and provide a proper perception of reality. SLAMEM uses ground truth entities to produce a realistic view of what could be captured by sensors within the simulation. This "view" is fed back into the simulation in the form of target tracks.

The Runtime Infrastructure and Communication

The RTI is the common component which the simulations use to communicate across the network. The RTI uses subscription spaces to divide simulation traffic so that federates only receive objects and interactions in which they are interested. For the DCEE, JFCOM is using RTI-s. By using RTI-s, we have been able to modify the methods of Data Distribution Management (DDM) (Helfinstine 2001) to allow for a much larger number of subscription spaces. For example, in a test completed in March, the city of Los Angeles alone was subdivided into 8,000 subscription spaces and the terrain was divided into over 100,000 spaces. In contrast, the Millennium Challenge 2002 federation was limited to approximately 1000 spaces (Ceranowicz 2002). This change was possible only by changing the communication protocol and using a routing application to perform management controls. All applications in the simulation learn about their routing by reading the Runtime Initialization Data (RID) file and communicating accordingly.

Building Software, Distribution and NFS

To maximize simulation time and minimize time needed for setup and configuration, we have found that having a single source for file distributions and builds is best. Troubleshooting applications started on an SPP is difficult. We do not have the option of redirecting output of possibly thousands of applications across the WAN. Therefore, we attempted to limit the number of variables as much as possible. We have a single administrator build and package the distribution and then distribute software tunneled through a secure shell (SSH).

Currently, we are configured to use a Network File System (NFS) for all applications and RTI files, but terrain information is stored on each node locally, because the network load of loading binaries is minimal and only occurs at startup. In contrast, the paging in of terrain can create a heavy load during execution.

Application Execution

Executing applications for a distributed simulation in an SPP environment is not simply a matter of submitting a batch job and having everything work. Prior to runtime there must be a process of gathering resources, creating a topology, distributing the Runtime Initialization Data, and preparing command lines.

To aid in performing these functions, we developed a set of programs which consists of a daemon to run on each node, a collector daemon to run on the head node of each cluster, and a controlling GUI to run at JFCOM. This system was designed to work in conjunction with a system that was already controlling local machines at JFCOM (See Figure 2).

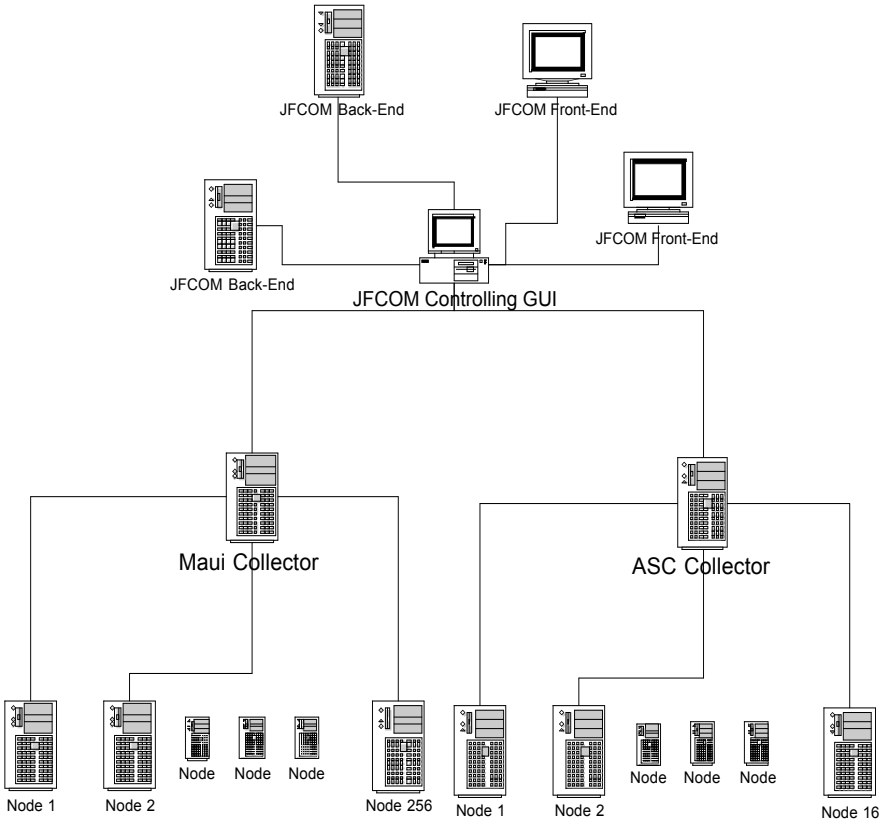

Figure 2 – GUI/Collector/Node Daemon Framework

Starting the Daemons

Generally speaking, each cluster has a job queue that manages resources and schedules job requests [and for the most part, SPP policy disallows interactive login shells.] Our approach is to start the collector on the head node and then submit the node daemons to the job queue (see Figure 3).

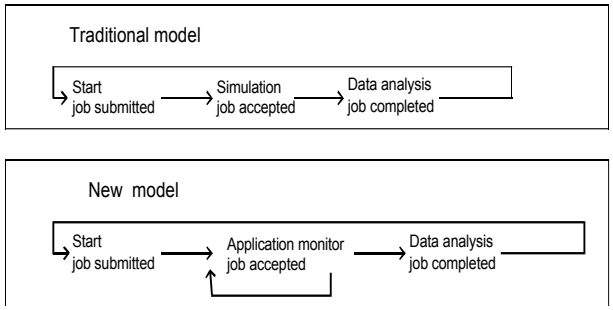

Figure 3 - Comparison of work flow between the traditional batch model and modified batch model.

Once the job is running, using the front-end, we can interactively start and stop applications. If a topology or configuration requires modification, the front-end can automate the workflow so the entire group of cluster applications start and stop with a single click.

New Way of Doing Business

Our methodology adheres to the cluster-job submission policy and provides a flexible interactive (and automated) interaction between the operator and the SPPs. This new approach warrants a brief discussion on this new way of doing business. Figure 3 compares the traditional method of interacting with clusters to the new approach. This approach provides real-time control interface that can often optimize resource utilization and provides (retains) a semi-automated model for rapid start/stop of applications.

Furthermore, this model is relatively fault tolerant. Despite previous efforts to make SPPs more fault tolerant (Squyres, 2000), the traditional model has some drawbacks. For example, if a node or an application dies, that node is lost. To restart applications on that node, users must submit new job submission requests. Using the interactive SPP model, operators can restart dead applications in real time, or even reconfigure the cluster topology, without having to start over from scratch.

Gathering Resources

Discovering resources on multiple SPP systems presented some problems. The first problem was trying to interact with machines on a disparate network. We opted to run a collector daemon on the head node that could be contacted by both applications on external networks and daemons that run on the internal nodes. All commands to nodes on a cluster would then be tunneled through the collector. This collector maintains a list of all nodes, and when requested by an approved external application, it provides a status of the nodes.

A second challenge that the SPP environment presented was the absence of a broadcast/multicast capability. This meant that to recognize existing machines on the cluster, the nodes would have to be configured with the IP address of the collector prior to execution. When the node daemons start up they attempt to connect to the collector to pass it their address information. Thereafter, the daemons send a heartbeat signal once per minute, while the collector gathers information on the node resources and waits for tasking from the external controlling application.

Within the heartbeat is information regarding free memory, CPU type, and load. This information is used to give the resources a rating that represents the expected simulation capabilities of the given node.

Abstracting Network Connectivity with Topology

Because broadcast messaging does not scale well as resources increase, RTI communication with the SPP is restricted to point-to-point (or a subset of broadcast, instead of global broadcast). Formal organization of the communication topology maps applications to resources. Two connectivity maps are currently being implemented and studied: (1) tree-based topology (Hellfinstine, 2001), and (2) mesh-based topology (Brunett, 1998). We have successfully tested the tree-based topology in current and past experiments, and we intend to pursue in a near future the implementation of the mesh-based topology.

Based on network load observation and metrics collected during test trials, we designed each topology instance. A topology is always conceived before simulation and is represented by a suite of perl scripts that generate RID files containing the connectivity maps.

Creating a Mass Launch File

Following the topology generation process, we know which application will be running on a given resource. The next step is to create a file to store the parameters specific to each application instance. This file generation is an automated process which produces a text output that can be read by the cluster controlling application.

Operators can then cut, copy, paste, and modify entries in the mass launch files as necessary. Multiple files can be generated to support multiple launch configurations.

JSAF back-ends require a special procedure. Each JSAF back-end is assigned a Persistent Object (PO) database number, which enables control from a front-end with the same database number. This number is usually shared between multiple back-ends to allow the entity load to be distributed among multiple machines. Grouping PO databases with topology groupings lessens traffic across the routing applications and keeps objects and interactions to a local set of machines.

Executing a Mass Launch

Once the Mass Launch File is completed, the system should be ready for launch. The operator of the controlling application then selects the tasks to be performed, enters the exercise name, selects the proper terrain and hits the launch button.

The collection of tasks is then sent to the corresponding collector, which in turn delegates to the proper nodes.

To handle the large number of tasks potentially given to the collector, a task queue was created for prioritization, future scheduling of tasks, and for the limiting of simultaneous connection threads to ten to prevent overwhelming the system. During a test in March, we were able to launch 240 applications on as many nodes within 60 seconds.

After receiving the application execution task from the collector, the node starts up the application and sends out a heartbeat. The collector receives the heartbeat which should indicate the application has started. At this point, the controlling GUI's operator can update the status to ensure all applications started up properly. As the applications start, they can be observed joining the federation by executing RTI print commands within the parser of any local federates.

Problems with Running on a Cluster

Typically when running simulations in a Local Area Network (LAN) environment, we run an application in the GNU Project Debugger (GDB) and direct the output to either a local monitor or a central monitoring station. This allows us to obtain stack traces and interact with any applications as necessary. Because SPP machines have no displays and redirecting a display across the WAN is not an option we are unable to easily monitor output. This adds difficulty to troubleshooting and difficulty in resolving configuration issues. We do log output to a file, but this is much less useful than an interactive debugging session.

Killing Apps / Restarting the Simulation

Once set up, the federate and routing applications can be started and stopped as necessary to support the needs of the federation. Stopping the simulations is accomplished in a similar manner to launching the simulations. The operator selects the tasks that were launched and simply presses a button to bring down the selected applications. As in the launch, tasking from the collector to the nodes is controlled to prevent overloading the systems. Operators can also select individual nodes to restart if desired. This option might be necessary if a specific application or group of applications become unresponsive.

SPP Resource Monitoring

The monitoring of SPP resources was recognized early on to be a necessity in any simulation support system. Specifically, we wanted to graph memory usage, load average, and network statistics on individual nodes. We needed to monitor these parameters in real time to analyze how changing parameters, code, topologies, and the many other variables associated with the simulation were affecting the individual machines.

We also wanted to ensure that any monitoring design did not create an unnecessary load. We decided to put monitoring capabilities into the same node daemon, collector, and application GUI framework that had been designed for the application execution system. This allowed the operator to query the resources for information by going through the collector

Analyzing Memory/CPU Utilization

We decided to embed the load average and memory usage information in the heartbeat from the node daemon to the collector. For these parameters, the once a minute sampling rate seemed sufficient and gave a good snapshot of the status of the nodes. We found that graphing the information in relation to time provided the most useful information by providing insight into trends.

We also wanted to ensure that any additional factors we wanted to graph or analyze could be added without much work. Efforts were made to generalize the graphing and data gathering capabilities so that additions could be made easily.

Analyzing Network Statistics

Network information was not embedded in the heartbeat because the once-a-minute reporting did not always suit our needs. Quite often, a five-second polling interval could catch spiking data that a sixty-second interval would not.

To accomplish this, we had the application query the collector with a list of nodes to be analyzed. The collector then queried the node daemons for network information which could then be used to create or update an existing graph. Using this method the operator can select any update rate he or she desires. Since this process generates much heavier load than obtaining memory or load information, we would typically limit the nodes we would graph to a small set.

Expected Problems and Bottlenecks

Each simulation we support has unique limitations. In benchmark testing, we observed that a JSAF using RTIs consumed about 20 KB of memory per entity subscribed or published. This meant that a JSAF application on a machine with 1 GB of RAM could be aware of about 40,000 entities before memory swapping occurred.

This number is not definitive due to a variety of other factors, including terrain paged in and other applications loaded.

Clutter only subscribes to some of the interactions and does not subscribe to remote objects at all. This means that memory becomes less of an issue and that the CPU becomes the primary limiting factor on number of entities the application can simulate. Figures 5 and 6 show CPU and memory information graphed from a machine on a mega-sim cluster node running a clutter application.

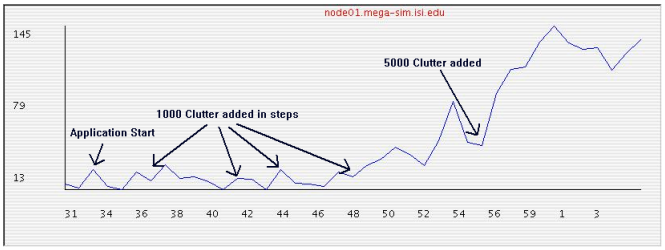

Figure 5 - Clutter CPU Load (Y axis) over time (X axis)

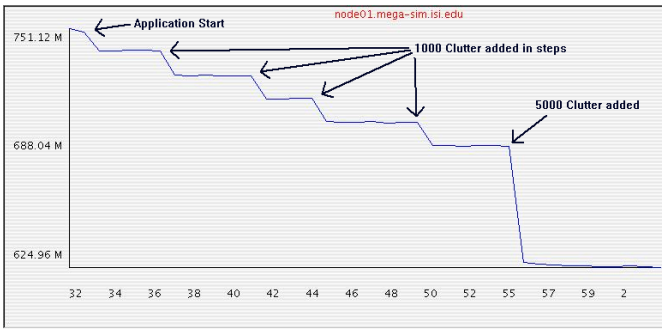

Figure 6 - Clutter Free Memory (Y axis) over time (X axis)

Routing applications on the head nodes of the clusters were the first location where network bottlenecks were expected to appear. When an application, or group of applications, over-subscribes to data that is then requested to go across the WAN, available network bandwidth can easily be exceeded. Figure 7 shows the network statistics on the head router on mega-sim. For the test, a JSAF at a remote site subscribed and unsubscribed to clutter on a cluster node. Graphs such as this can identify problems in DDM, machines, and networks.

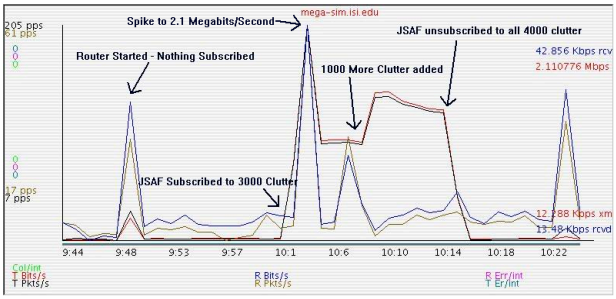

Figure 7 - Head router network statistics (Y axis) over time (X axis)

Although subscription spaces may be divided into very small areas, individual entities or groups of entities can still have subscription regions that overlap a large number of spaces, creating a large amount of traffic. For example, during a test in May 2003, we simulated approximately 100,000 clutter vehicles in the Los Angeles area on the Maui SPP. We then flew an F-16 (simulated at JFCOM) into the area. The F-16 had a subscription radius of approximately 30 nautical miles and subscribed to a large portion of the 8000 spaces in the area. This quickly overwhelmed both the local JSAF and pushed the network limits as well. While this type of effect was expected in this test, it demonstrated that a simulation operator can easily perform a seemingly insignificant action which will have catastrophic consequences on the federation. By adding monitoring capabilities to the routers we will better be able to catch these issues before they become a problem and help developers create better mechanisms for interest management.

Running a simulation with a very large number of entities and using dynamic subscription regions increases the risks. The potential to "blow out" any element of the federation exists and requires the support team to recognize and try to predict potential problems. Monitoring must inform the development team how changes in software, topologies, or network infrastructure, all machines throughout the simulation. Through diligent monitoring, methods to ensure the simulation delivers the most information possible while operating reliably can be discovered.

Security

To ensure that only individuals with proper authority are able to execute applications on the cluster from remote sites, we have examined two possible solutions.

SSL

Currently, all commands to the collector from the GUI application are executed through a Remote Method Invocation (RMI) connection via a Secure Socket Layer (SSL) (JSSE 2003). The collector is configured so that only clients with a private key (also known as the keystore) and the proper passphrase can connect. The keystore and passphrase are verified by a public key or truststore kept on each of the collectors. The use of SSL serves two functions: data encryption and authentication. SSL uses public key cryptography to provide authentication, while secret key cryptography and digital signatures ensure privacy and data integrity. The private/public key pair is generated prior to file distribution. The public key is then placed in the distribution, allowing the collector to authenticate connecting clients. The distribution of the private key must be controlled and only placed on machines which are designated to control the cluster.

Kerberos

The High Performance Computing (HPC) officer requires that SPP users have a Kerberos (Tung 1999) access card and that applications running on their clusters adhere to their security policy. We are currently studying options to ensure the experiments follow the strict HPC security guidelines, including:

1. Use of Kerberos tunnels between SPPs and operational centers which would authenticate and encrypt all communications.

2. Re-design the communication layer with the explicit use of the Kerberos library for point-to-point communications.

Future efforts include experiments to analyze the performance impact and validate both approaches.

Conclusion

We have attempted to design a system that gives a single administrator the capability to set up, execute, and monitor multiple simulation applications, on possibly thousands of machines, on disparate networks. We have also stressed simplicity and minimized the number of steps required to run the simulation. We believe that we have succeeded to a great degree.

The application execution system alone should save countless hours by simplifying a process that had been extremely obtrusive. By getting away from the traditional SPP batch model we have added the ability to bring applications up and down at will. Further, the monitoring capabilities we have added to the system should allow us to predict and recognize problems that may degrade the simulation. All the tools we have created are not simply nice to have, they are a necessity if we wish to run interactive simulations in an SPP environment.

Difficulties will continue to arise when dealing with systems that we do not own. These difficulties that must be addressed include: adhering to other organization's security rules, resource scheduling restrictions, and system management procedures. In addition, future simulation efforts, though dealing with systems that are complex in nature, should be simple as possible to use.

ACKNOWLEDGEMENTS

The authors would like to thank Andy Ceranowicz, Bill Helfinstine, Steve Bixler, Rae Dehncke, Jason Boyer, Phillip Amburn, Ken Hornstein, Nicholas Pellegrini, Gene Wagenbreth, Ke-Thia Yao, Dan M. Davis, Robert F. Lucas, George Thompson, and all the various site support personnel for their help in various aspects of this project.

REFERENCES

Burnett, S., Davis, D., Gottschalk, T., Messina, P., Kesselman, C., (1997), Implementing Distributed Synthetic Forces Simulations in Metacomputing Environments. Retrieved April 15, 2003 from ftp://ftp.globus.org/pub/globus/papers/sf-express.pdf

Burnett, S. and Gottschalk, T. (1998). "A Large-Scale metacomputing frame for the ModSAF real-time simulation." *Parallel Computing*. Elsevier, Amsterdam.

Ceranowicz, A., Torpey, M., Helfinstine, B., Evans, J., Hines, J., (2002), Reflections on Building the Joint Experimental Federation.

DREN webpage. http://www.hpcmo.hpc.mil/Htdocs/DREN/dren-def.html

Helfinstine, B., Wilbert, D., Torpey, M., Civinskas, W, (2001), Experiences with Data Distribution Management in Large-Scale Federations, Simulation Interoperability Workshop, 01F-SIW-032, Sept 2001.

JSSE Reference Guide http://java.sun.com/j2se/1.4.1/docs/guide/security/jsse/JSSERefGuide.html

Kuhl, F., Weatherly, R., and Dahmann, J., (1999), Creating Computer Simulation Systems: an Introduction to the High Level Architecture. Prentice Hall.

Squyres, J.M., Barrett, B., Lumsdaine, A. The System Services Interface (SSI) to LAM/MPI. Technical Report TR575, Indiana University, Computer Science Department.

Tung, Brian (1999). Kerberos: A Network Authentication System. Addison Wesley.