COLLEGE OF INFORMATION SCIENCES AND TECHNOLOGY
THE PENNSYLVANIA STATE UNIVERSITY

High Performance Computing for Agent-Based Cognitive Modeling

Jeremy M. Lothian

jlothian@psu.edu

25 February 2011

Phone +1 (814) 865-4453 Fax +1 (814) 865-5604

College of IST, Information Sciences and Technology Building, University Park, PA 16802

High Performance Computing for Agent-Based Cognitive Modeling

Jeremy M. Lothian
jlothian@psu.edu
College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802

25 February 2011

Abstract

The aim of this project is to evaluate a High Performance Computing (HPC) environment for running simulations involving a large number of intelligent agents. Simulations may benefit from separating the environment from the intelligent agents. This could allow for larger scale simulations, and different environments may alter the results of the simulation. A simulation environment was developed for agents to interact within. This environment and agents were tested using a standard computer server, and an attempt was made to use cluster computing resources to run the environment and agents on a larger scale. Difficulties in the account setup process, and technological limitations of the existing cluster environment lead to an unsuccessful test on the cluster. The communication method chosen (sockets) for the client-server interactions was not available on the cluster. However, tests in two single-server environments were successful. The outcome of this scenario suggests that more development is needed to address the portability of the communication model used for the client and server. Additionally, account creation procedures for the HPC environment may benefit from a streamlined process that addresses the rapid academic lifecycle of student researchers.

Acknowledgements

Preparation of this manuscript was partially sponsored by a grant from DTRA (HDTRA1-09-1-0054). The views and conclusions contained in this report are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government or the Pennsylvania State University.

Frank Ritter has provided useful comments, but incompleteness and inadequacies remain the fault of the author.

Table of Contents

1 Introduction

Many simulations involving intelligent agents are carried out with a single agent, or multiple agents working independently of one another. The environment that the agents work within is entirely self-contained inside the agent framework, and mostly conceptual rather than actually defined. For larger scale simulations involving multiple agents interacting, the usage of an environment that is separate from the agents themselves may increase the accuracy of the simulation, while additionally providing a platform for agent interactions, allowing the simulation to scale and support a much greater number of agents.

The goal of this project is to determine the impact of decoupling agents from their environment and provide information regarding the scaling capabilities of simulations in a High Performance Computing (HPC) environment.

2 Method

2.1 Experimental Overview

A series of three experiments (EX1-EX3) was designed, using different client/server setups, and combination of agent types. They were created to allow us to transition from a local development environment to a cluster environment, while maintaining asynchronous development of different systems. EX1 was a prototype environment, designed to simply evaluate the load capacity of the simulation environment and detect any initial design flaws. EX2 was a test-type environment, designed to emulate a cluster environment on a single machine to assist in development prior to getting access to the cluster. EX3 attempted to use the actual cluster environment, using systems and software developed during EX2.

2.2 Environment and Agent Setup

2.2.1 Simulation Environment – A-VIPER

Rather than code an entire simulation environment from scratch, a framework, NakedMUD, was chosen to provide a base which could be modified to suit our needs(Hollis, 2009). The choice of this framework was for two reasons. Firstly, a primary design decision of NakedMUD is that it provides only the framework itself, and does not define any complex environment properties or objects, which means that no code had to be removed to begin development. Secondly, NakedMUD is easy to extend, as it includes a scripting language (Python) interpreter embedded as part of the server code. Both of these factors are important, as the lifecycle of academic projects can be incredibly short, and rapid development is a necessity.

Alterations and additions were made to NakedMUD to accommodate agent interactions. These included a few additional commands, and the development of a virtual *area* for the agents to interact within. The area created was shaped as a simple ring, allowing an agent to move in a circular path and end up back where they started. Within the environment, an area consists of several *rooms*, which are discrete locations, connected to each other by *exits*, given in the cardinal and intercardinal directions (N,E,S,W;NE,SE,SW,NW). This implementation with modifications is being actively developed by Jeremiah Hiam, and is called the Agent-based Virtual Implementation of Plural Environment Representations (A-VIPER).

One important command that was added was a *log* command that creates a named log file and writes all actions taken by agents in A-VIPER to a time stamped log file. This allows us to track the agents' actions and interactions over time, separately from the cognitive functions that drive them.

In a cluster environment, the network IP Address that a given process runs on will not be known until the process is executing. Because of this, agents cannot be setup prior to execution with a specified IP Address. Additionally, jobs are generally heterogeneous in nature – that is they only run a single executable across multiple compute nodes.

To facilitate this, the MPI- A-VIPER -Runner was developed. This process runs via MPI (Message Passing Interface), which is a common system used on clusters. MPI runs multiple copies of the same executable, but orders the processes in a rank (from 0 to *N*) that is accessible inside the program. Using this rank, it is possible to make different executing copies of the same process provide different behavior.

The MPI- A-VIPER -Runner was setup to run A-VIPER as the Rank 0 process, and ACT-R agents as Ranks 1-*N*. MPI, as the name implies, also provides a communication system between the distributed processes. This was used to communicate the IP Address of A-VIPER to the agent processes, to allow for TCP/IP connections to be established properly. This is further detailed in Section 2.2.2.2, *EX2 and EX3 Agents - ACT-R*.

2.2.2.1 EX1 Agents-Shell Scripts

To test the basic functionality and scalability of A-VIPER on commodity server hardware, a prototype series of "agents" were created. These agents were generated simply as a list of commands to be sent to the server. They did not contain any ability to interact with the server beyond sending simple commands, or parse the response text they received. Each agent was run from a separately generated and unique script, containing five-thousand commands.

While these agents were not intelligent, they did exhibit two distinct sets of behaviors, which may be used in data analysis at a later time. The commands were generated randomly, but according to a set of simple probabilities. Since the commands were produced without any awareness of the environment, they include commands that may not be available during certain agent states (such as an agent attempting to move north, when no path north is available).

2.2.2.2 EX2 and EX3 Agents - ACT-R

A simple agent was developed by Jaehyon Paik using the LISP-based ACT-R cognitive framework. Unlike the EX1 prototype agents, the ACT-R agents were able to parse the response data from the server. For this reason, they were able to choose randomly from directions available in any particular room, rather than picking from a complete list of potential directions, which may not be available (such as with the EX1 agents). However, beyond picking appropriate directional exit movements, these agents were not given any further cognitive abilities. These agents were used for both EX2 and EX3.

The ACT-R agent required two major changes from the initial software model provided by Paik to facilitate working in an HPC environment with MPI. Firstly, since the IP Address of the server could potentially change for every experiment run, and would be unknown until runtime, each agent needed the IP Address of the A-VIPER server to be provided dynamically during runtime to the agent. Secondly, the user-login for the server had to be unique for each agent, or A-VIPER would detect each subsequent connection as a "reconnect", disconnecting the current user, and connecting the new one to the same account.

Both of these issues were overcome by generating pieces of the agent LISP code during runtime. During startup, the processes allocated to run agents (process ranks 1-*N*)) would wait for the A-VIPER process to communicate its IP Address via MPI. They would take this information, together with their process rank, and generate pieces of the agent's script files (agent_server*N*.lisp and agent_rank*N*.lisp), and a file that loaded the pieces in the correct order, agent_run*N*.lisp (see Figure 1, arrows indicate file inclusion), and then proceed to execute this script.

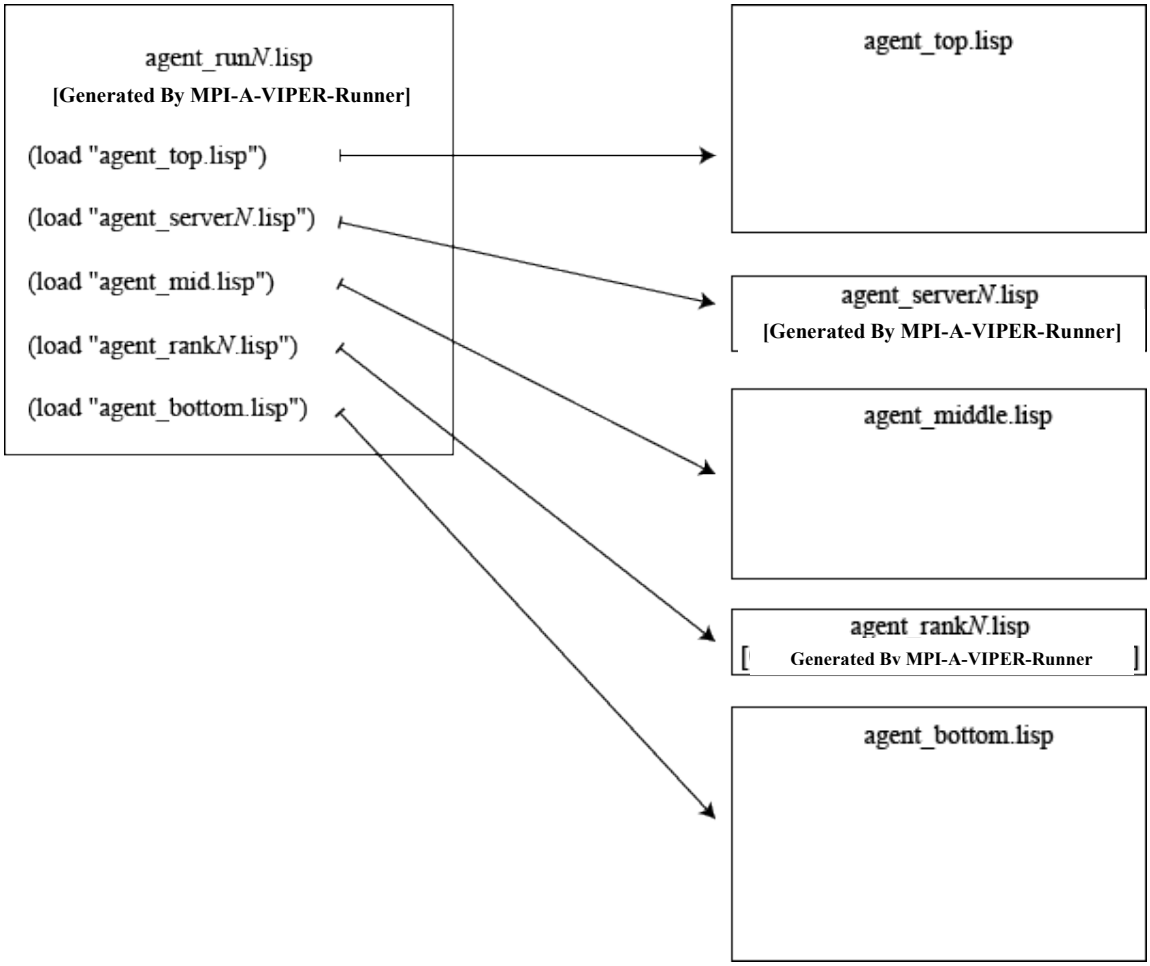

Figure 1 – Model of Dynamic Agent Construction.

2.3 Experiment Environment Setup

Each experiment required a different client/server environment setup. These are detailed in the following sections, and presented in Figure 2. In Figure 2, conceptual isolated environments are represented as dashed lines. These are generally separate machines, but in the case of EX3, also represent the cluster environment itself. Arrows indicate communication pathways between the different software and systems.

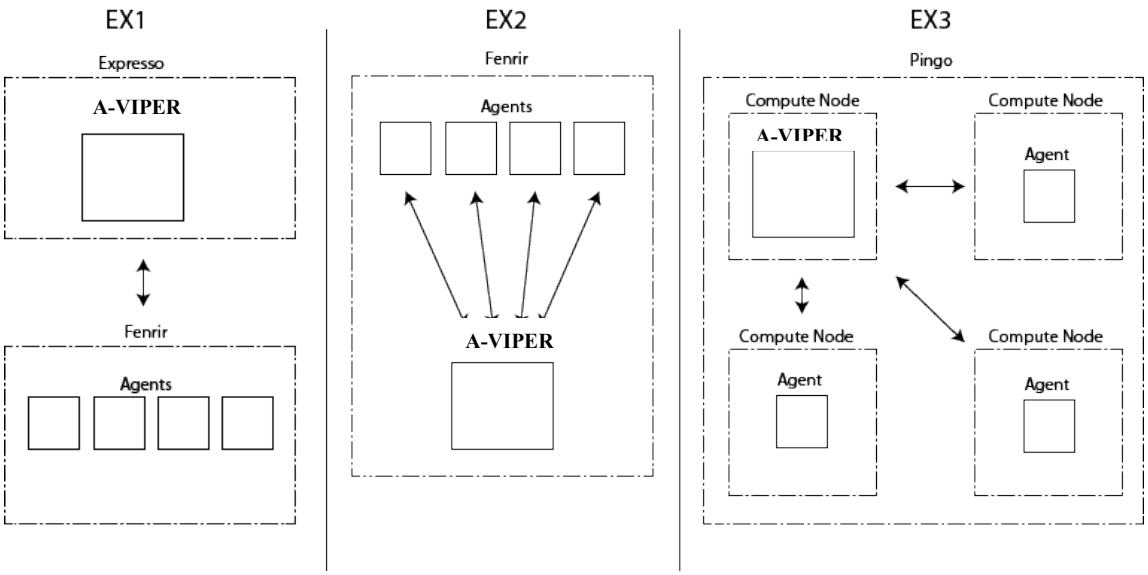

Figure 2. Environment Configurations by Experiment.

2.3.1 Experiment 1 (EX1) Setup

EX1 was run using two different computers. The first, *Expresso*, was responsible for running A-VIPER. Expresso is an Apple Power Macintosh G5, running Apple OS-X Leopard with two G5 2.0 Ghz PowerPC 970 processors and 1GB of RAM. Access to this machine was provided by the Applied Cognitive Science Lab at The Pennsylvania State University.

The second system, *Fenrir*, was a VMWare Workstation Virtual Machine (VM) running the Redhat Fedora Core 14 Linux distribution. The VM was allocated the resources of 8 logical processor cores and 6GB of RAM. The VM host system (*Heimdall*) is a custom built computer with an Intel Core i7 930 processor running at 2.8 GHz. This is a quad-core processor, with Hyper Threading, enabling 8 logical processor cores. Additionally, it utilizes Intel's VT Extension technology, enabling virtual machines to make better use of the available hardware. Only half of the available 12 GB of RAM was allocated to the VM, to prevent starvation of resources to the host system. Fenrir was used to run the EX1 agents.

Expresso and Fenrir were connected over the open Internet, with a packet latency that fluctuated between 25ms and 35ms. Expresso was connected through Penn State, and Fenrir utilized a consumer-grade broadband connection of 25Mb downstream, 5Mb upstream. For the communications between these two servers, bandwidth would not have been a bottleneck.

2.3.2 Experiment 2 (EX2) Setup

EX2 represents a prototype "cluster" environment running on a single local machine. The Fenrir VM was used to host the environment. Versions of MPI (MPICH2), and LISP (Clozure LISP) that are available on the cluster were installed on Fenrir in an attempt to mirror the software environment of the cluster in a single-server setup. The MPI-A-VIPER-Runner was then used to execute A-VIPER and the ACT-R agents, in a similar method as they would execute on the actual cluster. The main difference in this case would be that rather than processes each running on a separate node, they shared processor cores on a single node.

2.3.3 Experiment 3 (EX3) Setup

The high performance computing resources were provided by the Arctic Region Supercomputing Center (ARSC), which is a joint project between the University of Alaska and the Department of Defense (DoD). They were made available for this project as part of the Defense Threat Reduction Agency (DTRA) group's ongoing efforts to provide defense-related simulations. ARSC provided us with access to Pingo, a Cray XT-5 cluster with 3456 compute cores across 432 individual nodes.

The operating system on Pingo is called the Cray Linux Environment, which is a heavily modified version of the SuSE Linux distribution. It provides several frameworks and packages critical to the usage of the cluster environment. However, it does not provide an implementation of LISP, which is required to run ACT-R agents. This was quickly installed during the account setup process as our requirements were reviewed, through some communication with the support staff regarding the implementations of LISP that are capable of running ACT-R.

Because of the nature of the ARSC resources, being provided in part by the DoD, the process for receiving an access account requires several steps and confirmation to ensure security. Additionally, it required certificate confirmation of several security training lessons provided by the DoD.

3 Results

3.1 Experiment 1 (EX1)

Initial runs of EX1 yielded a critical problem with A-VIPER and the agents. It was not capable of accepting multiple connections simultaneously, and rather than queue additional connections, they were rejected, causing the remaining agent instructions to fail. This was adjusted for by inserting a delay between each agent connection, allowing A-VIPER to accept connections one at a time.

This solution introduced a few additional problems. Firstly, there was no way to guarantee that all the agents would be running at any given time during the experiment. Secondly, if all the agents started in the same room, several movements at once would be sent to all the other agents in a large block of text, which would not accurately indicate the load of many agents moving across the area. To handle this, all of the agents would execute 200 random movements immediately after connecting. Then, all of the agents except the last one (1 to N-1) would await a synchronization command from A-VIPER sent from the final agent (N). While no exact time was recorded, this initialization and synchronization took less than a minute for 20 agents and was not included as part of the total running time of the simulation.

This synchronization command was written to the log file, so that the log file could be parsed to exclude all activity before the simulation was synchronized. The simulation took 20 agents 4 minutes and 37 seconds to complete, generating a total of 115,596 successful commands, corresponding to processing roughly 418 commands per second.

Increasing the number of agents resulted in noticeably slower speeds, and tests that were not run through to completion. It was not clear if the speed reduction was due to processing limitations on Expresso, or latency caused by the network connection. However, both of these considerations would not be issues during EX2. While the speed was reduced, A-VIPER kept accepting connections, and processing commands, indicating that it was stable enough to support a large number of agents interacting.

3.2 Experiment 2 (EX2)

Unfortunately, time did not allow synchronization and random initialization to be added to the ACT-R agents prior to EX2. However, with longer runtimes, and appropriate delays, it was still possible to roughly estimate the load on the server, since the delay time was caused primarily by loading the LISP interpreter and ACT-R.

With a higher processing capacity available on Fenrir, and no network latency issues, it was possible to run many more agents. For EX2, 80 ACT-R agents were connected to A-VIPER. They ran for a total of 1 minute and 58 seconds, after being set to run for an ACT-R modal time of 2 minutes. In total, they executed 120,456 commands, for an average of 1,021 commands per second.

In comparison to EX1, this was 4 times the number of agents, executing commands more than twice as fast. A run of 120 agents was attempted, but several of the agents failed to execute properly due to an error with MPI that indicated resource limitations of the machine (Fenrir). This seems to indicate that A-VIPER is limited primarily by the computing resources available and network latency, and should scale quite well to the HPC environment.

One problem that occurred during this experiment was that logging was not automatically activated when A-VIPER was run. To overcome this, a delay was inserted between A-VIPER being started, and the agents executed. During this delay, a user logged into A-VIPER and manually turned on the logging function. This is a temporary solution, as users will not have access to A-VIPER in a real HPC environment during a simulation. A better solution would probably to start the A-VIPER server with logging turned on by default.

3.3 Experiment 3 (EX3)

3.3.1 Account Setup

The account setup and creation process is designed to provide a single point of contact, the Service/Agency Approval Authority (S/AAA), for all communications during account setup. The S/AAA assists potential users in understanding the process, and guidance in providing the appropriate information on required forms. In practice, the S/AAA may employ an assistant to handle many of the basic inquiries and process.

In our case, email communications with our support were often unanswered, and telephone conversations seemed to indicate that individuals involved were handling many tasks simultaneously, and only recently been given the job of handling setup of HPC accounts. This led to confusion regarding a particular form, "*Section III*". The purpose of this form is to acknowledge receipt of RSA SecurID cards, and other credentials, so that a user's account can be activated, allowing them to login to the cluster. This is a useful step in a secure process, and potentially stops individuals from intercepting credentials for an active account. Our support contact informed us that we must fill out the Section III to receive our credentials. This created a Catch-22 situation where we were required to sign a form to get the credentials that we required to acknowledge having prior to signing the form.

It seems that this problem stemmed from a process that is designed around a user being on-site when filling out the application. In this case, they would be handed their SecurID card, and credentials, and immediately fill out the Section III, acknowledging this receipt. In our case, since we were not on-site, our credentials would have to be mailed to us.

The ARSC was aware of this problem, and provided an alternate Section III form, which was identical except for the omission of acknowledging credential receipt. However, our point of contact was not aware of this, and our group spent considerable time trying to explain the problem, before contacting the ARSC directly, which solved the problem quickly.

3.3.2 Project Setup

Because the environment (A-VIPER) utilized a cross-platform framework, and was developed in a BSD[1] environment, completing a basic compile of the source code on Pingo was relatively simple. However, compiling it in a way appropriate to run on cluster compute nodes was quite a bit more complex.

Generally, when compiling a C program from source code, a user can choose one of two methods to link in shared library resources. They can be linked statically, where the shared resources are included in the final compiled binary executable, or dynamically, where they are linked to the executable during run-time. The first method (static) creates a larger executable, which may not reflect changes in the system libraries as they are updated. The second method (dynamic, or often *shared*), creates a smaller executable that requires access to the shared library at runtime, and will load the version of the library on the system that may have been updated.

Dynamic linking is generally preferred for most binary executables in a Linux environment, as it keeps all of the executables using a shared library up to date with the same version of that library system-wide. Additionally, it requires considerably less disk space, and allows for sharing of memory resources within the library.

However, this is not generally appropriate for a cluster environment. Most programs designed to run on a cluster use a single copy of an executable, running concurrently dozens or hundreds of times with a different part of the dataset. Running hundreds of copies of the same executable with dynamic linking would cause each copy of that executable to request the shared resources at nearly the exact same time, generally over some kind of network connection, rather than locally. This could be a massive performance drain to any other tasks running on the cluster, and because of the network communication involved, could even affect tasks if they were running on different nodes. Additionally, compute nodes generally do not have a very feature-rich environment, and do not include many of the shared libraries available to the system.

Initial builds of A-VIPER using dynamic linking compiled acceptably on the Pingo login nodes, but failed to execute on compute nodes because the shared libraries were not available. With some minor adjustments to the build process, the project compiled using static linking without errors and was available to begin running test jobs. However, the embedded Python used by A-VIPER would not run correctly when included as a static library. If our case, we were fortunate that the ARSC/CCAC Helpdesk was able to provide a work around (copying the shared libraries to a work directory that was available to the compute nodes).

With this fix, it was possible to run A-VIPER on the compute nodes. However, connecting to A-VIPER remained an issue. To connect to A-VIPER, we needed to know the IP Address of the compute node that was currently running it. A simple test program was setup to collect the network information of the executing node and display it. This lead to the discovery that the compute nodes did not have normal network interfaces, and TCP/IP connections would not be possible. The ARSC/CCAC helpdesk informed us that this should be possible on the new cluster,

[1] BSD – Berkley Software Distribution is a type of UNIX operating system, similar to Linux. In this case, we were using the Darwin derivative, which is the underlying architecture of Apple OS-X.

Chugach, but due to the time available, we were not able to move the project to Chugach and test prior to this report.

4 Discussion and Conclusions

4.1 HPC

4.1.1 Programming Language Availability

In our case, LISP, required to run ACT-R, was not already installed on ARSC's Pingo cluster. Fortunately, ARSC was able to accommodate installation of LISP for us. This was not without problems. In general, they are only allowed to install certain GNU[2] approved software packages in the environment. The first LISP they found (GNU GCL) was not capable of running ACT-R.

While the second LISP (Clozure LISP) functioned acceptably, the process did raise some interesting concerns. Table 1 represents a brief, abbreviated list of many of the available cognitive architectures, and the programming language they are implemented in (Samsonovich, 2010) ("Comparative Table of Cognitive Architectures,"). Where more than on language is listed, the sources did not always indicate if they were separate implementations, or if parts of the architecture were written in different languages.

Immediately, it is noticeable that many of the cognitive architectures are implemented in LISP or Java. Java may be available in many cases, but was not in ours (ARSC did have it installed on a separate Sun cluster), but is not a guaranteed resource on different clusters. LISP, however, is generally not available, and would require installation on nearly any HPC cluster. While ARSC was willing to work with us towards the installation of LISP, this may not always be the case in other HPC environments.

While LISP implementations are generally noted that they are compatible with Common LISP, they may not implement *all* of Common Lisp, as we found with GNU GCL. Depending on the features of Common LISP used by the cognitive architecture, this could lead to an architecture being incompatible on certain LISP implementations. Additionally, there are many features implemented as core technologies, or library functions in languages such as Java or C++, which are not specified by Common Lisp. For instance, different implementations of LISP have different methods (or no methods) for handling command line arguments, and network communications. This introduces an additional risk that major components written in an architecture may have to be rewritten for a different version of LISP in an HPC environment.

A partial solution can be found in Common LISP libraries that are developed independently of a specific LISP implementation. However, for the more advanced libraries, installation can be non-trivial at best, requiring a user to track down several dependency chains of libraries required to run the library they wish to use. Package managers exist, but often favor a specific implementation of LISP, and also would be another component that would require installation in the HPC environment. Furthermore, some features, by their nature, cannot, or are not implemented in libraries (such as using command line arguments).

[2] GNU – Recursive acronym for GNU's Not Unix. In this context, GNU Projects, which implement free versions of several tools supported by the Free Software Foundation (FSF).

Table 1 – Abbreviated list of Cognitive Architectures by Programming Language("Comparative Table of Cognitive Architectures").

Architecture	Programming Language
3D/RCS	C++
ACT-R	LISP
BECCA	Java/MATLAB
CERA-CRANIUM	Java
CogPrime	C++/Java
EPIC	LISP/C++(EPIC-X)
FORR	LISP/C++
GLAIR	LISP
GMU BICA	Matlab/Python/LISP/Java
HTM	NuPIC
Leabra	C++
NARS	Java/Prolog
NEXTING	C++/Java/Perl/Prolog
POLYSCHEME	Java
Recommendation Architecture (RA)	SmallTalk
REM	LISP/Java
Soar	C/Java

4.1.2 A-VIPER Ease-Of-Use Features

A-VIPER was developed as a means to quickly and effectively implement features in an environment required for different kinds of agents to interact. However, many of the features that made this platform attractive also made it difficult to work with in an HPC environment. The inclusion of Python embedded into the server requires that Python be available in the HPC environment. Fortunately, Python is typically used for HPC, and is readily available. That said, the workaround that we used to enable embedded python using dynamic linking may not be an option on other HPC clusters.

The use of sockets over TCP/IP granted us a relatively easy way to allow communication between the environment and agents. However, on many clusters, Ethernet is not used for the communication topology; instead the network is usually implemented with faster interconnection methods, such as Infiniband, which do not utilize TCP/IP. The general solution to this problem is to use a system such as MPI to manage communications, which is covered in section 4.1.3, *The Need for Decoupled Communication*.

4.1.3 The Need for Decoupled Communication

As evidenced by our difficulties utilizing TCP/IP on Pingo, the communication method is a major risk factor for any project intended to run in an HPC setup. Sockets provide a perfect example of a coupled communication method. Sockets require IP Addresses for connections over TCP/IP, and make very few, if any, intelligent decisions about how the connection should be setup[3].

This is one of the reasons that MPI was developed. MPI provides an interface and communication system that allows the programmer to concentrate on building the messages, while not needing to be very considerate to the method of communication. MPI can make connections using TCP/IP, Infiniband, shared memory, or other methods, depending what is available on the executing system.

MPI works using a model where a single executable is used to perform different pieces of a task based on the process rank that the executable has while running multiple copies. Communication is available between the different copies of the executable that are running. This is generally an excellent way to divide up a computationally heavy task among many worker processes.

In our case, MPI is not an appropriate solution to implement our server and agents for a few reasons. Firstly, A-VIPER and the ACT-R agents are written in different programming languages and it would not be possible to implement them both in a single executable. Secondly, even if they were in the same language (which might be the case for some simulations), the simulation environment, and agent architecture are generally performing very different, and resource intensive tasks. Implementing them both in the same executable could lead to a considerable waste in resources, and unnecessary complexity for development.

In EX2, we utilized MPI in a way that allowed us to run a light-weight MPI executable (MPI-A-VIPER-Runner), to execute both A-VIPER and the agents. This method results in MPI being unavailable as a communication method to the child processes, since MPI requires a very specific system and compilation to function. As a result of this, and because our agents and server are different programing languages, we are still in need of a system that serves the same function as MPI, but provides the features using a different method.

One common system for this, used in enterprise applications, is Common Object Request Broker Architecture (CORBA). CORBA provides methods for programming language and platform independent communication between applications("CORBA FAQ,"). CORBA works by allowing developers to specify an object using a standardized Interface Definition Language (IDL), and mapping the IDL to data types available in each language that implements a CORBA library. This is similar in nature to how the Simple Object Access Protocol (SOAP) works utilizing XML with web services.

The key differences are that SOAP generally only works via RPC and HTTP (both protocols are built on top of TCP/IP), and that SOAP provides connections directly between a client and a server. With CORBA, communication may take place using TCP/IP, but architectures can specify other communication protocols, and rather than being a direct connection between client and server, CORBA utilizes an Object Request Broker (ORB) that facilitates communicating between different services and components.

[3] The operating system may provide some slightly intelligent behavior, such as short-circuiting a request that should be sent to a router when it finds that the requested IP Address is local to the machine.

While potentially communication protocol independent, the additional inclusion of the ORB makes CORBA inappropriate for HPC usage. This introduces an entirely new service that must be run fulltime on the cluster, rather than just a library or protocol, which makes it much more unlikely that CORBA would be installed for any given project. Built for enterprise business scenarios, CORBA is also not very light-weight. Setting up a service and client to use CORBA can take extensive time, and does not facilitate the rapid development that is often necessary in an academic environment.

Finally, there is the High Level Architecture (HLA) utilizing a Run Time Infrastructure (RTI)(Dahmann, Fujimoto, & Weatherly, 1997). HLA was developed specifically for simulations, communication between simulation components, and communication between interactive users and the simulation. It was developed by the Department of Defense for the purpose of providing a framework for defense simulations. For this reason, HLA and the RTI are very strictly defined, and require a certain methodology to implement in a simulation environment. In order for a simulation to make use of HLA, the environment and agents must be designed around utilizing it, and the objects it provides.

Generally, this means that many currently complete, or in development cognitive agents would not be able to make use of HLA without intense development and refactoring. Furthermore, it may be the case that the structure of HLA either does not provide a component needed by the agent to communicate properly, or changes the nature of the communication, altering the simulation results.

4.1.4 Rapid deployment and testing

While large-scale HPC resources are a welcome addition to agent-based modeling, the process of gaining access to the resources, and time required to configure the environment, may not always be fast enough to match the rapid lifecycle of an academic project – in some cases, only lasting a few months. For this reason, there is a need for more promptly available resources. One optimal solution would be to streamline the account creation process for the HPC environment, but this may not be a viable option.

A more likely option would be to make use of available computers already installed in the academic environment. The Bootable Cluster CD (BCCD) allows a lab of computers to be turned into a cluster by booting from a Linux-based CD, with some minor configuration. BCCD provides many of the same facilities available on larger scale HPC environments, such as MPI and multiple compute nodes. Initial setup can take some time, as there are certain configuration issues that must be addressed (such as if a computer's Ethernet MAC address is tied to a particular static IP address, or not). However, the CD can be customized to automatically populate some of this data, resulting in a cluster environment that can be running in the time it takes to insert a CD into each machine, and boot them.

There are other options that do not require commandeering the entire resources of each computer. Condor Clusters make computer resources available only while the computer is not in use. Software is installed on the machine that is capable of running scheduled tasks. When the computer is not in use by a normal user, the node is available as a compute node. There are a few drawbacks of using Condor. Firstly, programs must be specifically designed to work on a Condor Cluster. Secondly, because of the nature of how the cluster works, a job divided among different nodes may not execute each node process at the same time, and therefore jobs that require communication between processes are not possible.

A final suggestion would be to install a communication protocol, such as MPI directly on the computers. By itself, this would not be a viable option, as MPI cannot detect if a user is using a

machine. However, with a custom monitoring process, and a specialized scheduler, these resources could be allocated appropriately. The caveat being that a computer would have to be locked from allowing user logins while running an MPI job, to facilitate proper performance. The risk to users could be minimized by the job scheduler only running jobs during non-peak usage hours (such as during the middle of the night). While this does limit the availability of the cluster, it would make considerable use of otherwise unused resources. It would also allow the flexibility of not requiring the individual running the experiment to be present (such as would be required with BCCD), and still allow node-to-node communication (which is not possible on Condor).

4.2 Insights

4.2.1 Local Testing Versus HPC Testing

Local testing allowed us to uncover many potential problems implementing the simulation and agents. However, it was not possible to immediately foresee all of the potential issues with running on Pingo. While we were able to setup many of the same conditions that were similar to the cluster, it was hardly a substitute for actually having the cluster available for testing. Some of the problems we observed were mitigated by available documentation (such as noticing that LISP was not available on the software list for Pingo). However, others, like the lack of TCP/IP could not easily have been discovered until we had access to the machines, or without prior experience dealing directly with these kinds of environments.

4.2.2 Complexity of Project

With each component required to run both A-VIPER and the agents, the project became more difficult to implement in the HPC environment. For example, for each separate programming language used, the likelihood of not being able to run the project increased. This also includes the libraries used by each component. A-VIPER requires several system libraries available to run, include libraries for compression, encryption, math functions, python functions, and others.

Each of these introduces an additional risk that the resource may not be available in any particular HPC environment, and that A-VIPER or ACT-R may not be possible to run. Even if the resources are available, or partially available, it may take considerable development to make use of them on a compute node, reducing the amount of time available to run meaningful simulations and increasing the risk that a particular experiment may not be completed in the time allotted.

4.3 Future Work

4.3.1 A-VIPER Refactor

To decrease the risks mentioned in sections 4.1.2 and 4.2.2, A-VIPER should undergo some refactoring. Many of the libraries used may not be required. For instance, encryption is used primarily for encrypting account passwords, which is not needed for a simulation running on a private server. Likewise, compression is used to compress the data stream between A-VIPER and the agents. However, the bandwidth available will generally be much greater than the amount of communication required by A-VIPER, and the compression requires proper implementation of the Telnet protocol to activate, which the agents do not implement. For this reason, it may be possible to completely remove the Telnet implementation within A-VIPER for a performance gain.

4.3.2 Flexible Solution to Communication

To reduce the risk associated with using sockets, a flexible middleware solution should be implemented that decouples the communication method used between the simulation and the agents. This implementation would be similar in spirit to MPI, but allow for processes that are not run from the same executable, enabling task of a more heterogonous nature to be run using multiple nodes. Primarily, it should allow different communication models to be implemented in an underlying architecture that is separate from the data that needs to be communicated.

CORBA (with IDL) and SOAP (with WSDL), provide models for constructing communication channels that can provide type-safe, structured communication between different processes. As an improvement over SOAP specifically, it would be necessary to allow both the client, and server, to have fulltime two-way communication. Utilizing a variant of service contracts like IDL and WSDL, interfaces could be automatically generated in a variety of programming languages, keeping the programmer from having to spend time to implement the data parsing and communication method.

However, unlike CORBA, this solution should be implemented as accessible as programming language libraries, and set of multiplatform applications with limited dependencies, rather than require extensive middleware. This would enable the communication to be rapidly integrated into the server and agents, while not requiring the design of the agents and server to be heavily modified to meet the needs of the communication method.

4.3.3 Future Agent Simulations

We were unable to perform any complex modeling during these experiments, leaving considerable questions unanswered. The purpose of providing an environment for agent interactions is not solely to allow the simulation to scale to support large number of agents, but also to determine how having an environment affects what, and how the agents learn.

The nature of having a predefined area dictates that the area have a specific shape, with different movement options available at different locations internal to the area. One path of future research should look at how the shape of the area affects the interactions of the agents, and how they learn in the environment. For instance, how do interactions differ in a circular area to those in a complex area that includes dead-ends? Are different kinds of agents, or those that have different knowledge able to navigate one type of area better than the other?

In fact, the interactions of different types of agents could result in a separate line of research. Take, for example, the notion of three kinds of agents, with different movement behaviors. The first moves randomly, choosing from the directions available to it at each room. The second marches back and forth along a predefined set of rooms. The third follows a distinct path, but only in one direction. Each of these agents will have a different probability of experiencing different events within the area. This will affect the memories they acquire, and knowledge they obtain, which will subsequently affect the actions that they take in the future.

4.4 Conclusion

While EX3 was never successfully run, the process of running EX1 and EX2, and the preparation for EX3 lead to many useful insights and directions for future work. When dealing with an HPC situation, there are several risk factors that should be considered, which may limit an experimenter's ability to operate in a timely manner. Likewise, the design of the simulation may need to take into account the possibility of having HPC resources available, during the initial setup phase. The provided suggestions should help minimize these risks in the future, and give

examples where future work could help improve the design and flexibility of implemented solutions.

5 References

Comparative Table of Cognitive Architectures. Retrieved 2010-12-17, from http://bicasociety.org/cogarch/architectures.htm

CORBA FAQ. Retrieved 2010-12-17, from http://www.omg.org/gettingstarted/corbafaq.htm

Dahmann, J. S., Fujimoto, R. M., & Weatherly, R. M. (1997). *The Department of Defense High Level Architecture*.

Hollis, G. (2009). NakedMud: Content-less MUD Engine. Retrieved 02/20/2010, from http://homepages.uc.edu/~hollisgf/nakedmud.html

Samsonovich, A. V. (2010). *Toward a Unified Catalog of Implemented Cognitive Architectures* Paper presented at the Biologically Inspired Cognitive Architectures 2010.