Midwife: CPU cluster load distribution of Virtual Agent AIs

Ovi Chris Rouly
Department of Computational Social Science
George Mason University
Fairfax, Virginia, USA
e-mail: maelzel@ieee.org

Abstract — **This describes a CPU cluster load distribution algorithm that may be helpful when simulating crowds of artificially intelligent Virtual Agents. The cluster load distribution algorithm was instantiated in Java code, demonstrated successful execution on a small cluster of inexpensive CPUs, and supported the simulation of Virtual Agents within a readily available, open source, Virtual World client-server toolset. The client-server toolset used was the network-enabled, multi-user virtual environment(s) of OpenSimulator and OpenMetaverse operating together. Although these Virtual World tools are well known, this experiment was unique in that the psycho-socially significant small-group oriented crowds of Virtual Agents in the simulation enjoyed interacting, individual, cognitive, affective, and behavioral Agent controllers executing entirely client-side. In this Artificial Life experiment, each CPU in a scalable CPU cluster controlled up to fifteen, individual, C#-based Agent controllers. This argues for and demonstrates how disembodied computational Agencies can be executed in distal computer environments when network-coupled to convenient Host embodiments in a Virtual World. There are implications in general here for a paradigm shift in Virtual Agents/Virtual Worlds modeling and in specific for positive computational load distribution of artificial intelligence in Virtual Agents inhabiting complex, emergent, crowd-scene models just by using more effective tools and modeling methodologies.**

Keywords - Virtual Agents; Virtual Worlds; mobile agent; cluster load distribution; multi-agent modeling; Artificial Life; OpenSimulator; OpenMetaverse; Second Life.

I. INTRODUCTION

In the very broadest sense, the current work extends research from the domain of distributed artificial intelligence (consider Huhns, M., [9]; and Weiss, G., [17]). However, here we are concerned with a more focused application. We are concerned with a simulation and modeling support methodology that computes multiple, distributed, artificially intelligent agents (specifically Virtual Agencies) within a scalable cluster of networked CPUs and we expect to see these Virtual Agencies ultimately embodied (Hosted) in some distal Virtual World. This is an example of "cloud computed mobile agents" (so eloquently described in Satoh [14]) and predicted over a decade ago by Luck and Aylett [10], Thalmann [15], Mendex, et al [13], and long since implemented by Herrero, et al [8], and others. Each of these described or used many of the various enabling and separable components inherent to Virtual World and Virtual Agent research. Their works make the current work more understandable.

The problem addressed here exists in the context of scalable communities of adaptive, social, "large computational process" Virtual Agents that experience individual simulated "birth" and "death" within a Virtual World. The research question is thus: Can an automated software system be constructed that will: 1) work with OpenSimulator/OpenMetaverse, 2) be coded in Java and work effectively with C#, 3) negotiate with a scalable cluster of CPUs to determine which of them should host the next Virtual Agent requiring simulated "birth," and 4) can this system dynamically task the least "busy" CPU within the cluster to "birth" (instantiate) the next Agent? Midwife demonstrated that the answer was yes. Moreover, this work contributes relative to the state of the art when it is considered Midwife is the only OpenSimulator / OpenMetaverse-capable, open-sourced, small cluster load balancing toolset available for immediate public download.

Many (for example Cao, et al [5]) have offered computationally "agentized" grid computing load-balancing systems. Such systems are purposed to balance the computational load of a generic compute-cluster. In addition, such load-balancing systems are sometimes reliant on complicated algorithms (e.g., the Genetic Algorithm in the case of Cao, et al) for scheduling functionality. The foregoing features are in contrast with the simplicity of the Midwife recursive load-balancing proxy auction system and its explicit purpose to support Virtual Agents. In Midwife, Virtual Agents are the consumers of computational load balancing effects. They are not the algorithmic scheduling components.

The Midwife algorithm (and its Java instantiation) can be compared to a set of Remote Procedure Calls (RPCs). Midwife is explicitly described as facilitating the occupation of a Virtual World by scalably large numbers of artificially intelligent Virtual Agents. In the current work, one finds a constructive proof demonstrating that a Virtual Agency can be usefully separated from its Host embodiment in a Virtual Environment and it confirms how the mobile agent paradigm may lend itself favorably to crowd simulations in pursuits as diverse as science or industry, Computational Social Science, or entertainment and games. Download Midwife source code from: http://css.gmu.edu/papers/Midwife_(rev_0.1).zip or http://www.maelzel.com/source/Midwife_(rev_0.1).zip.

II. METHOD

The experimental method involved a familiar problem motivating a novel technological solution. The need came from a requirement to support an ongoing computational social science experiment where multiple autonomous Virtual Agents occupied an empirically grounded, non-deterministic, Virtual World simulation. Created was a technological solution capable of supporting the simulation of that virtual world (a bounded virtual habitat) wherein the simulated carrying capacity of the virtual environment was sufficient to sustain several dozen anthropomorphic Virtual Agents over long periods of simulated time.

In the multi-agent, colony-oriented experiment motivating this work each Virtual Agent is capable of demonstrating autonomous, emergent, Artificial Life. OpenSimulator 0.7.5 (www.opensimulator.org) was used to construct the Virtual World. The Virtual Agents were constructed of original C# code built upon C# code libraries provided by OpenMetaverse 0.9.1 (www .openmetaverse. org). This ongoing research is attempting to create self-aware automata embodied as Virtual Agents. Therein anthropomorphic (proto-human) Virtual Agents are explicitly instantiated with a capacity to breed, forage, mingle, age, die, and all-the-while use embedded artificial genetic structures to cybernetically steer long-term simulated biological and cognitive (psycho-social/spatial) existential results. The need-based requirement for a software like Midwife emerged after some time when it became apparent that more effective tools and methods were required if the experiment was to continue plausibly as the requirements of colony membership numbers continued to grow.

A. *Motivation*

In general, this experiment exemplifies one extreme boundary condition of multi-agent modeling wherein extant Virtual Agents must “birth” (or instantiate) entirely new Virtual Agent “infants” into an operating Virtual World. In particular, whenever a simulated "mother" Agent within a colony “gives birth" to an "infant" Agent it is only the spatial and temporal demands of the respective Agency ("mother" or "infant") that must be coupled to (or with) one specific, respective vehicular Host, i.e., an avatar embodiment. There was no, is no, computational necessity or simulation-value inferred or added by having the "mother," "infant," or Host avatar computed on the same CPU. It was only, is only, necessary that the particulars of spatial and temporal colocation of the Virtual Agent Agency and the vehicular Host (avatar) embodiment coincide such that a human viewer is satisfied with the renderable result. In other words, as shown in the referenced experiment, even when a “mother” Virtual Agent “births"(instantiates) a new “infant” Virtual Agent (instance) it is only the new avatar embodiment that must emerge in the Virtual World at a time and place consistent with the constraints of the simulation, e.g., colocation in simulated time and simulated space. As a programmatic mechanism, the Midwife algorithm and the Java code instantiated for this experiment was shown capable of facilitating the instantiation (or "birthing") of Virtual Agent Agencies across the physically distal nodes of a CPU cluster. It did this while allowing the Virtual World simulation to give a human user (with a rendering viewer) the visually satisfying sense that social, spatial, and temporal colocation properties associated with a "mother" and "infant" pair in an anticipated world environment were taking place. Typically, although with some exceptions, this sort of client-side avatar control has been relegated to the pejorative domain of “test ‘bots” and is not exploited to its full capacity. Here however, it has been elevated to its fuller end.

B. *Technology*

The Midwife Java software was developed on a single-board, 64-bit Intel Core2Quad microprocessor operating at 2.33 GHz. That CPU was an early model Zotac m-ITX device equipped with 4GB RAM and board-based Nvidia (brand) 9300 graphics acceleration. Three operating systems (OS) were successfully used with Midwife. These included Microsoft (MS) Windows XP, MS Windows 7, and MS Windows 8. The MS Windows XP OS is the current CPU cluster environment due solely to quantity pricing constraints. Development took place mostly under Windows 8. A Java Integrated Development Environment (IDE) was provided by jGrasp version 2.0.0_04. The Java compiler was Oracle (www.oracle.com) 1.7.0_17, verification testing took place in Java SE Runtime Environment build 1.7.0_02-b02 with client Virtual Machine (VM) build 23.7-b01, and native mixed mode support. During development, the Zotac internal 127.000.000.001 loopback address was targeted. During cluster operations, system addresses 010.000.000.010 through 010.000.000.016 were routinely targeted.

The current operational hardware environment for Midwife is a cluster of four Intel m-ITX motherboards arranged as a stack of Ethernet enabled devices interconnected by high-speed switch. The network interconnection star topology speed is switch-selected 100Base-TX. Typical data rates are under 20 Mb/s at the server. The Intel m-ITX motherboards, each respectively, have one Intel Core2Quad microprocessor operating at 2.33 GHz. Each motherboard in the cluster runs as an independent microcomputer system, or CPU. To complete the multi-user virtual environment (MUVE), the CPU cluster containing the Virtual Agent agencies was connected across the LAN and to an Intel 5520 server running an OpenSimulator Virtual World simulation configured as 16 spatially contiguous, Hypergrid-Standalone, heavy processes each modeling one square kilometer of digital terrain data. This OpenSimulator Virtual World simulation contained the Virtual Agent host embodiments motivating the current work. The OpenSimulator Virtual World simulation was briefly described above. Most of the properties of that Virtual Agent experiment are extraneous to the needs and scope of this paper. The operation of the OpenSimulator product itself is, of course, described in sufficient detail on its own website by its own designers (see www.openSimulator.org).

III. ALGORITHM

The top-level program (code) in Midwife is a Java Class of the same name. Actually, Midwife is quite simple. It uses a multi-Class, object-oriented architectural design that incorporates discrete methods, procedures, and functions. The algorithm can be favorably described as a network-centric machine that counts the instances of Virtual Agent Agencies within the purview of itself, is aware of the count of Virtual Agencies within the purview of each similar machine on the network, and by sharing those count data with other Midwife machines on the network can act to balance the total computational load of Virtual Agent Agencies executing on the network by either decisively instantiating a new Virtual Agent Agency locally or by transferring that responsibility to another Midwife machine.

Figure 1 shows a block diagram depiction of the Midwife algorithm and its relationship to a local area network (LAN).

What follows now is a detailed *Design Description Overview (DDO)* of the Java instantiation. Source code from http://css.gmu.edu/papers/Midwife_(rev_0.1).zip is available or at http://www.maelzel.com/source/Midwife_(rev_0.1).zip.

A. DDO – Midwife.class

Midwife is the top-level program code (and Class) in the load-leveling Java instantiation. It is a step-wise iterated algorithm that establishes three timer-driven (interrupt-driven) code blocks and several important sub functions before exiting. Midwife sets the stage for the BirthRoom (a threaded Class explained below) to operate within the framework of independently firing timers each acting to continuously update and share available network and system information.

At program start the MS OS version is determined. Then, step one in the process is for the Midwife to identify the Internet Protocol Address (IPA) of its own host Ethernet adapter. This is done with a call to findLocalIPA. After doing so, the Class-visible variable myIPA is set. From then on the Midwife knows where and who it is in the CPU cluster network. Step two involves the Midwife instance determining the IPAs of any other CPUs in the cluster. (It is naively assumed the private network is closed and that Midwife is running on any CPU visible in the cluster.) This is done with a call to findClusterIPAs. After doing so, the Class-visible Array List ipaSubnet is made ready and this Midwife knows who, if any, the other active IPAs in the cluster are. The next step, step three, is intended for user comfort only, i.e., it is a side effect. This step reports the hosting CPU cluster subnet (as a human readable list) to a Graphical User Interface (GUI). Step four involves instantiating the central code elements of the BirthRoom class. It is from BirthRoom that a Midwife can provide "birthing" arbitrage, network load calculations, and support network interface utilities throughout the several processes of new Virtual Agent Agency instantiation.

Immediately after creating the BirthRoom thread, the Midwife assesses the count of active Agencies under its own local care. Clearly, since at this point this Midwife has only just begun to operate, there will be no active Agencies to discover and the current count of zero is reported to the GUI. The Class-visible variable activeAgents (the count of local Agencies) is now set.

The final steps involve sequentially starting three independent timer-driven interrupts. Each timer exists on an independent thread and will fire persistently and periodically until the entire Midwife program ends. Figure 2 illustrates by name and temporally interspersed nature of the timers. The timers are timerAgentList, timerIPAList, and timerMWLoads. Respectively, their purposes are:

- timerAgentList (shown in red) causes the reassessment of, and GUI display of, the current count of Virtual Agent Agencies under local care,
- timerMWLoads (shown in green) uses the "others" list (described below in the text) as a basis to sequentially ask all other Midwives in the hosting cluster to report their own individual, current, local Agency loading, and
- timerIPAList (shown in blue) periodically updates the hosting cluster IPA list (this is held as the Array List "others" within each Midwife).

As soon as the timers and the BirthRoom Class thread have been started, the initial Midwife process is free to leave

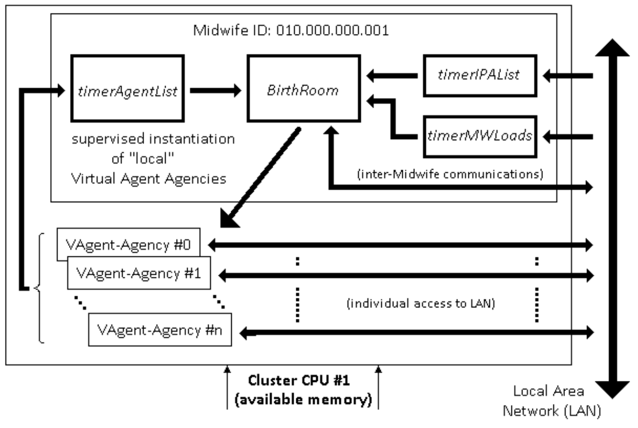
Midwife ID: 010.000.000.001
timerAgentList
BirthRoom
timerIPAList
timerMWLoads
supervised instantiation of "local" Virtual Agent Agencies
(inter-Midwife communications)
VAgent-Agency #0
VAgent-Agency #1
VAgent-Agency #n
(individual access to LAN)
Cluster CPU #1 (available memory)
Local Area Network (LAN)

Figure 1. Information flow as a Midwife communicates with others over a local area network.

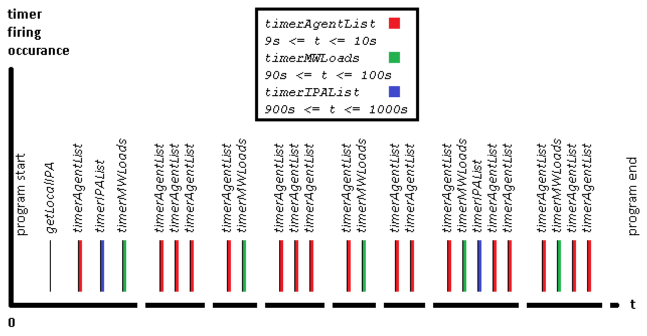

Figure 2. Asynchronous pattern of timer-driven interrupts in Midwife. The length of "t" is abbreviated.

execution scope. As it leaves focus, it leaves behind the three timers to continue firing and the BirthRoom thread to negotiate with the other Midwives within the CPU cluster for the rights and responsibilities to "birth" (to instantiate) new Agencies. Clearly, the timers will become more and more asynchronously aligned in their execution order with respect to their firing pattern over time since their period(s) are first established using randomized, real-valued periods and second they are never re-synchronized after they are started. This "randomized," asynchronous effect, is intentional. Its teleology is to continuously redistribute the computational load seen by each CPU and the entire CPU network.

B. *DDO – Console.class*

The Class, Console, produces a side effect: a GUI. This GUI informs the user about the current working status of the Midwife program and its processes.

The window constructed by Console is a small, non-resizable box with one display panel and two selection buttons. All network and ping reactive, i.e., capable of responding to a port 53 Domain Name Service (DNS) ping, Internet Protocol Addresses (IPA) in the host cluster are displayed in the window in numerically increasing order. During Midwife controlled operations the buttons are labeled *Freeze Expansion* and *Terminate* or alternately *Permit Expansion* and *Terminate*. The default start condition is disabled (Frozen) and thus *Permit Expansion* is shown. The buttons provide mutually exclusive functions. Figure 3 shows the Midwife GUI window in operation over a five CPU cluster. The window in the console can only show four IPAs at a time.

The button labeled *Freeze Expansion* stops (or alternately starts) the ability of this Midwife to create new Agents. When selected (when *Freeze* is selected) the label changes to *Permit Expansion*. When *Permit* is selected, the label changes back to *Freeze Expansion*. This method allows the user to freeze (or start) the local Midwife functionality. A count of the Agents under the active control of this Midwife

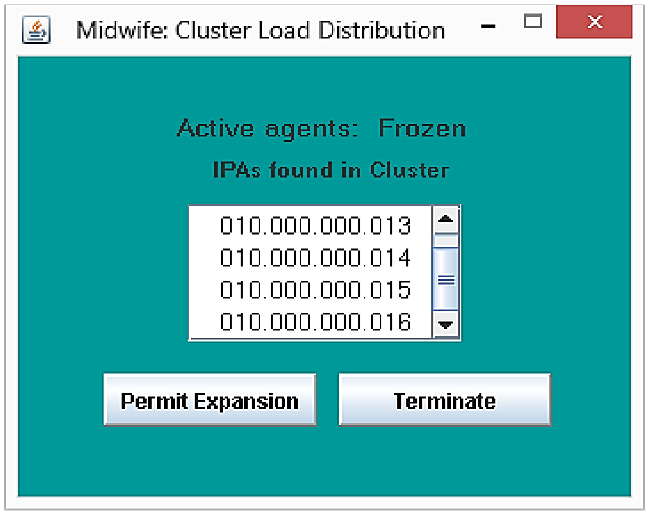

Figure 3. Screen capture of a Midwife console window operating over a system of five machines in a four CPU Midwife cluster.

is displayed next to the status label.

Terminate, the other button, ends this local Midwife operation but does not halt the Agents on the local CPU. This important capability is twice verified with the user for correctness before it terminates the Midwife. If a Midwife is started on a CPU in a system already in operations -even one having one or more Agents in existence- it will enter with no errors and begin operations as just described.

C. *DDO – GetLocalIPA.class*

The Class, GetLocalIPA, provides a network interface sub-routine. The sub-routine momentarily establishes a network socket, determines the default IPA associated with that socket (effectively the Midwife host CPU network adapter interface address) then stores the particulars for external accessor pickup. As soon as the task is completed and the data stored for accessor pickup, all task-execution memory used is returned to the OS. Server.class provides additional networking utilities.

D. *DDO – GetLocalIPA.class*

The Class, GetSubnetIPAs, provides important functions to the Midwife network interface. Its purpose is to discover the active set of IPAs on the CPU subnet. It uses naïve network connectivity observation to dynamically build a list. Once built, that list is available to external accessor pickup. Server.class provides additional networking utilities.

The Class is capable of scouring the 253 subnet network addresses adjoining the base IPA. By default, the address range is 1 to 254 on the span xxx.xxx.xxx.001 to xxx.xxx.xxx.254. It uses an MS OS specific call to PING.EXE to interrogate the space. This specific "ping" command checks for an active (and DNS port 53 responsive) IPA. Once "ping" is given to the OS, the OS returns a text-based response that can be captured and scoured for certain key feature string values confirming that the IPA is in use. Currently, the code assumes the OS is MS versions XP, 7, or 8. This methodology allows the OS return data to be parsed, and searched for the key string. The buffered input returned by the OS is passed between methods in GetSubnetIPAs and will ultimately produce a Boolean true if the key feature (a known string of characters) can be found. Else, a Boolean false is registered and the 1 of n IPAs searched are decided to not contain an active CPU.

Program startup command line switches /lo=n and or /hi=m, inclusive, can be used to steer the search somewhat in the event a known, smaller cluster set is predicted. Methods are provided to parse the input arguments from the command line to locate and return the value of the IPA value arguments. This attempt to recover and then return the IPA address low range value from the console input argument string and the IPA address high range value is clamped by internal defaults to 1 and 254, respectively.

E. *DDO – BirthRoom.class*

BirthRoom is the core component of Midwife. It provides new Agency instantiations ("birthing"), instantiation responsibility auction arbitrage and support, cluster load and loading calculation and prediction, and instantiates a private copy of the network Server and MsgInterface Classes.

The BirthRoom directs its copies of the Server and MsgInterface Classes to listen to network traffic for requests to start new Virtual Agent Agencies. Requests for new Agencies may come from one of three sources. The first is an external "Nursery" program. This program is (or any similar program could be) used to instantiate a primary seed group of Virtual Agents at the beginning of (or perhaps during) a simulation. Another source is the other Midwives themselves. They are empowered to direct each other to instantiate new Agencies in order to balance the distributed network load. The last source is from a local Agency needing to "birth" (to instantiate) an offspring (an independent, new) Agency. Figure 4 shows the inter-Midwife network communications message list.

As described in Midwife.class, a timer-driven interrupt starts a set of routines that provide the BirthRoom accessor access to updated copies of the Agency load status of every other Midwife on the subnet and to a copy of the Array List variable Midwife "others" (the other proven active IPAs) in the subnet. These data are provided via double-buffered, synchronized transfers. BirthRoom will continue to execute until it is pulled out of memory by the Console component or an external hardware reset. Agencies, those controllers-of-the-Hosts in the Virtual World, are called into OS instance by the Midwife using an external OS executable currently named "Agent.exe". This executable is variable controlled and could just as easily be a selection made from a list of several Agency types needed in the simulation. When the executable is called it is accompanied by a parameter list of Virtual Agent in-world requirements like instantiation location, avatar sex and implicit pedigree (conventions used in the Artificial Life experiment described earlier), instantiation with or without prejudice, and a few other experiment specific, parameter defined items, etc.

Depending on the message received by the Midwife from the MsgInterface, a proxyAuction for the responsibility to instantiate new Agencies may be started. This is associated

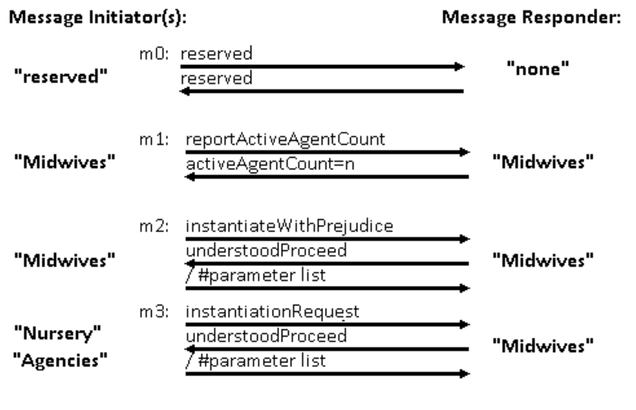

Figure 4. Direction flow and content of inter-Midwife network communications messaging.

with an instantiationRequest. Alternatively, a message may arrive as a demand to instantiate an Agency. This is associated with an instantiationWithPrejudice. These two concepts are now described.

The proxy auction within Midwife is a system intended to evaluate and assign the responsibility to instantiate the new Agency. The auction method (code) is itself recursive. This assignment process is at the heart of the load distribution process that is Midwife. Each Midwife is capable of holding an entire auction internally and independently. Effectively, the auction is an absentee auction process undertaken by the current Midwife where all eligible bidders for the responsibility to "birth" a new Agency, including the current Midwife, participate by proxy. The proxy token is a number that serves as a surrogate for the last known Agency loading value held by each remote Midwife in the cluster, respectively. That is, since the local Midwife knows what all Midwife Agency loads were 9 - 10 seconds ago (including its own value), the local Midwife can hold a fair auction (and place bids on the behalf of the other Midwives and itself) based on the respective local and surrogate loading values.

The auction is biased in favor of those Midwives having the least Agency loads. A lesser loaded Midwife is allowed to bid more aggressively for the opportunity to host a new Agency. Theoretically, the Midwife with the least Agency load will always win the right to host the new Agency. Tied bids are broken by random number selection between Midwives having the least current load in a recursive runoff auction. Once an auction begins new data surrogates are ignored should they arrive during an auction. Only the latency induced by shared data updates between Midwives can skew the accuracy of an auction. And, since time delays between shared data updates is variable selected, these too can be adjusted to tune the final system. Timer interval values of 30 to 90 (+5/-0 seconds) for interrupt timerMWLoads have been tested successfully and shown to produce usable results.

If the winner of the auction to instantiate the new Agency is not the current Midwife then a call to instantiate the new Agency is transmitted to the winning Midwife with an explicit command to start the new Agency with prejudice. When that other Midwife receives the call to instantiate the new Agency, the message flag signifies resolution by further, local auction process is not allowed. The message communicated is instantiateWithPrejudice. Once received, the other Midwife must instantiate the new Agency immediately. However, if the local Midwife wins the auction, then it must instantiate the new Agency on the local CPU.

Calls to "birth" a new Agency (with instantiationRequest) from either the "Nursery" or an existing Agency are passed to the Midwife in a slightly different manner. Calls from either the "Nursery" or an existing (and local) Agency are handled without prejudice and both arrive with a full parameter argument list. These requests to "give birth" are handled by the proxy auction process, as just described.

Whenever a Midwife receives a “request” to instantiate an Agency, the BirthRoom begins to simulate the proxy auction using the recursive method proxyAuction. If a Midwife receives a “demand” for instantiation (instantiateWithPrejudice), the method startAnAgent is called along with the respective parameter list of Agent particulars earlier described.

F. DDO – Server.class

Server.Class provides Midwife with several simple networking utilities. For example, the class has methods capable of quickly and safely instantiating sockets (ServerSockets and simple Sockets), testing them, and evaluating them. It provides execution exception handling away from the main program execution path and thus "de-clutters" the readability of the more central elements of the Midwife source code and its components. The InetAddress based protocols used by Midwife employ the 32-bit addressing format.

For convenience, Server also provides methods for network related variables service. These methods concern themselves with the manipulation of formatted string products depicting an IPA with interspersed "dots." For example, "000.111.222.123," a string depicting 4 terms each with 3 numeric characters, can be manipulated here to derive the subnet address of "123." An individual Midwife, in turn, uses this, as it associates itself with, and to, a Port. For example, Port=1123 can, and will, be assembled from the foregoing basic string components. Each Midwife is thus known by this exact IPA and Port schema.

Also from within server, the MsgInterface is instantiated as a child process. Although MsgInterface is discussed separately, it is worth noting that MsgInterface is the final step in the communications chain between BirthRoom and all external programs. Finally, Server.Class has a simple parameter steered millisecond timer for its own private use.

G. DDO – MsgInterface.class

The MsgInterface component transmits and receives network communications that support inter-Midwife, Nursery-to-Midwife, and Agency-to-Midwife network traffic. MsgInterface.Class instantiates two local (and distinct) IO network streams. The first is used for inter-midwife Agency load status messaging and a second for Agency instantiation ("birthing") requests and demands. Because these two IO streams are local to MsgInterface, the MsgInterface object also has local control to flush (and close) data in its network buffers associated with the streams, respectively.

MsgInterface is a singular child process of the Server.Class method and it maintains accessor methods for inter-Midwife message retrieval and transmission. Among those accessor support methods is a tool, getArguments that finds and returns the string arguments embedded in the parameter list describing new avatar location, avatar sex, and implicit pedigree requirements of the experiment in progress.

During load-balancing, it is imperative that each Midwife knows the loading status of each of the other Midwives in the CPU cluster. A method named getLoadingStatus takes care of this function. It communicates with the Midwives on the subnet and directs a request to each Midwife (including itself) to reveal the current, respective, Agency loading status. IP addresses on the subnet that have returned a positive response to a ping but that are not hosting a Midwife are excluded from further interrogation during load-balancing. These data too are accessor visible.

The MsgInterface main execution thread (in a method named Main) responds to incoming (inter-Midwife) traffic requests and demands to instantiate ("birth") new Agencies. Main reads the network through a method named receiveBuffer and attempts to identify the data. A helper method, readBufferConversion, is used to identify the data received as being composed of type String or StringBuilder. Then, Main guarantees conversion of the data received to type String and execution continues using a CONSTANT matching conditional tree.

Each section of the tree corresponds to one valid inter-Midwife messaging transaction pair. Figure 4 depicts the messaging schema. Contingent upon which branch of the conditional tree fires, the associated helper methods and accessor variables are activated or loaded, respectively.

This ends the DDO – Design Description Overview.

IV. RESULTS

The system under test had four cluster-dedicated CPUs. More nodes are possible. Three other machines were also on the network and were either setup explicitly for code development purposes or, because of graphics capabilities, were operated as Virtual Agent-Virtual World observers.

During the test, Virtual Agent instantiations occurred at a rate of approximately a one “birth” per minute for a period of just over one hour. Virtual Agent loads, plus one Midwife per board, demonstrated continuous (individual) microprocessor loading of 65%-75% as a maximum under full Agent accommodation, i.e., all 60 in the simulation. Midwife execution accounted for around one percent of the load per CPU. Each 32-bit (4GB RAM) CPU showed roughly +2GB of RAM usage under full agent accommodation. During early tests Midwife distributed 60 agent loads as 15, 14, 16, and 15 over the four CPUs. It was believed load-recognition and balance-lag trouble was the reason for the +/- 1 agent loading split. Lag was adjusted by parameter, the test restarted, and a 15 agent per CPU balanced resulted. The cost of improving balance was 1-2% additional network traffic for inter-Midwife interrogations activities. Overall, network loads typically lay beneath 20Mbs. “Migration of running agents” (between nodes) is not practical for this research due to accumulated “state memory” within each agent and the design of the AIs.

Limit testing involved an attempt to execute 80 Virtual Agents in our OpenSimulator Virtual World. The size of the agent control codes (AIs per cluster CPU) limited the number of agents under test. More agents with smaller code footprints would have allowed for a larger test cohort on the same system. Thus, results here describe a smaller test of 60 Virtual Agents. Identical agent control algorithms were used for the test. Each agent executed simultaneously and independently on one of the CPUs in the four board cluster.

V. DISCUSSION

In 2001, Adobbati, et al [1] described a game engine based Virtual Environment wherein human controlled agent-avatars and non-player character (NPC) avatars could interact in real-time. Their toolset grew out of the game engine Unreal Tournament. This MUVE, Unreal Tournament, was designed to allow mixed client-side controllers to enter a Virtual World simulation from anywhere a network socket connection could be generated. As an early adopter of this distributed type of Virtual World interface and its respective (IPA) protocols, game engines like Unreal Tournament laid the foundation for many of the MUVEs that followed. Then too, it implicitly created a need for some type of simplified mobile agent support for Virtual Agency load distribution (especially in the case of behaviorally rich AIs) and a need for Virtual Environment interface and protocol standardization.

The Midwife program is a Java-coded, multi-agent model, CPU cluster, load distribution software for Virtual Agents that enjoys the mobile agent paradigm. This paper advocates a paradigm shift in Virtual Agents/Virtual Worlds modeling towards the mobile agent paradigm and a modest change towards positive computational load distribution of artificial intelligence in Virtual Agents inhabiting complex, emergent, crowd-scene models. Midwife does not offer any new Virtual Environment interface and protocol standardization. That was offered in 1996 by the introduction of the IEEE Foundation for Intelligent Physical Agents (FIPA) standard.

Although, it was developed to support a MUVE simulation within an OpenSimulator/OpenMetaverse framework, the basis algorithm of Midwife should be extensible to any similar MUVE. The algorithm is premised on the concept that Virtual Agents are disembodied computational Agencies coupled to convenient Host embodiments or representations where, in many cases, the Host may be an anthropomorphic avatar. In fact, in a system where avatars explicitly serve as the vehicular Hosts to such Agencies, this symbiotic relationship need not place demands of computational colocation on either the Agencies or the Host representations. Amo, et al [2] implied as much when they observed that, “the Agency is proposed as an entity that controls the agents' life cycle and the interactions among agents” (p. 109). The Agency need not be inseparable from its vehicular Host. The requirement of computational colocation of Virtual Agent Agency with its Host is artificial or worse, toolset-artifactual. The Host and the Agency of a Virtual Agent do not have to reside on the same CPU unless the virtual modeling environment is extensibility challenged.

In 2007, Barella, et al [4] recommended a prototype software solution that dealt with an extensible and scalable differentiation of Agent control and was well within the domain of the mobile agent topic. Their proposal dealt with, “… in a separated way the visualization and intelligence modules …” (p. 532) that exchanged the control signals of, and in, a Virtual Environment. They recognized the clear value of taking a “distributed approach” to the control of Agencies and embodied Hosts (or objects) in a Virtual World. In fact, they took steps to develop prototype software demonstrating their solution. Coincidentally, their solution recommended adherence to an IEEE standardization effort explicitly intended for the control of Virtual Agents. The standard, called FIPA (mentioned earlier) was begun in 1996 in Switzerland. It remains to be seen if the Barella software called “JGOMAS” will find wide appeal.

In contrast, Midwife does not deal with the control or generation of behavior signaling passing between Agency and Host in the Virtual Environment. Midwife is only responsible for starting a Virtual Agency. Moreover, Midwife has been shown to operate over the well-known OpenSimulator and OpenMetaverse framework and not just a semi-custom Virtual Environment.

In 2010, Aversa, et al [3] recommended a system consistent with the mobile agent paradigm. Their system intends to improve the computational diversification of Agency control and thus facilitate improvements in other simulation and rendering tasks. They explicitly advocate the distribution of Virtual Agency computations across a compute-cluster (or cloud) using a “cloud-on-GRID” algorithm, they call “Cloud Agency.” The mechanism claims to increase the efficiency of Agency computations by distributing them to any platform on a network with spare clock-cycles. It likely would. However, the Aversa proposal also takes research control of Agency distribution out of the hands of the Virtual Agent researcher and puts it into the hands of the corporate cloud service vendor. In contrast, the Midwife algorithm always starts by leaving control in the hands of the researcher. Depending on the nature of the experiment and the data the experiment produces, the tradeoffs “Cloud Agency” requires may become too costly for reasons not involving money.

Clearly the concept of controlling client-side Agencies having rich behaviors is not new (consider Grimaldo [7]; and Ullrich [16]). However, even while they did not elevate the issue of computational load distribution into their discussions nor did they raise the issues associable with large-scale simulations as did NPSNET in the Macedonia Ph.D. dissertation from 1995 [11]. Both of these subjects, rich client-side NPC behavior and large-scale simulations could profit from the mobile agent paradigm and a process for CPU cluster load distribution of Virtual Agent AIs.

Indeed, there may not be occasions where it is possible for an Agency and its Host to be collocated on the same processor due to properties of the experiment or model. An example of this might involve a crowd scene wherein NPC Virtual Agents are expected to interact with human controlled agent-avatars in a riot or disaster relief simulation involving tens or even hundreds of artificially intelligent NPCs and human controlled agent-avatars. This realization frees us to suggest that if more than one processor, or CPU, is available in a compute-cluster, or perhaps cloud computing environment, then some form of load distribution and Agency-identifier message passing system would be helpful in maintaining an efficient use of the computational resources during a Virtual World simulation in terms of the number of simulated Virtual Agents.

For example, if it were required that 1,000 or 10,000 mixed human and NPC avatars inhabit a real-time, non-deterministically developing MUVE then, a mechanism to distribute the computational loading presented by the Agencies would clearly be useful. Durupinar [6] might be a case in point where the research was concerned with modeling crowds (“Audiences to mobs”) of psychologically motivated Virtual Agents. Midwife was created to provide just such a mechanism albeit demonstrated here on a quantitatively smaller, small-group scale. Midwife is a CPU cluster, load distribution mechanism intended for use with Virtual Agents inhabiting a Virtual World. It was explicitly developed for its utility and to demonstrate its effectiveness as an extensible prototype. It is an application of the mobile agent paradigm in Virtual Agent research.

VI. SUMMARY

This paper disclosed a detailed Design Description Overview of the methods and components that are the Java instantiation of the Midwife algorithm. The basis algorithm of Midwife was said to involve a network-centric machine that counts the instances of Virtual Agent Agencies within the purview of itself, is aware of the count of Virtual Agencies within the purview of each similar machine on the network, and by sharing those count data with other Midwife machines on the network can act to balance the total computational load of Virtual Agent Agencies executing on the network by either decisively instantiating a new Virtual Agent Agency locally or by transferring that responsibility to another Midwife machine. Historical and existing works related to, or similar to, the Midwife mechanism were discussed. The most relevant of these works must be identified clearly as the Aversa, et al [3] “Cloud Agency” where an eloquent description of cloud computed mobile agents was provided by Satoh [14] and circumscribed the general area of performance.

In summary, it is believed that in the context of similar Artificial Life or large scale Computational Social Science experiments involving “virtual human societies” (consider Thalmann [15]); those where emotionally, cognitively, and motor-effector behaviorally rich simulations involving large numbers of mixed human controlled and NPC controlled avatars must inhabit a real-time, non-deterministically developing MUVE then, a scalable CPU cluster load distribution mechanism would be useful.

Midwife was created to provide just such a mechanism albeit demonstrated here on a quantitatively smaller, small-group scale. Midwife is a CPU cluster, load distribution system intended for use with Virtual Agents inhabiting a Virtual World. It was explicitly developed for its utility and to demonstrate its effectiveness as an extensible prototype. It is an application of the mobile agent paradigm in Virtual Agent research.

ACKNOWLEDGMENT

Thanks to the folks at OpenSimulator, OpenMetaverse, and Linden Research – Second Life (Trade Mark).

REFERENCES

[1] Adobbati, R., Marshall, A. N., Scholer, A., Tejada, S., Kaminka, G., Schaffer, S., and Sollitto, C. 2001. "Gamebots: A 3d virtual world test-bed for multi-agent research." Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS. Vol. 5. Montreal, Canada. Amo, F. Alonso, et al. 2001. "Intelligent virtual agent societies on the internet." Intelligent Virtual Agents. Springer Berlin Heidelberg.

[2] Amo, F. A., Velasco, F. F, Gomez, G. L., Jimenez, J. P. R., and Camino, F. J. S. 2001. "Intelligent virtual agent societies on the internet." In Proceedings of the 3rd International Workshop on Intelligent Virtual Agents (IVA). Lecture Notes in Artificial Intelligence, vol. 2190. Springer-Verlag, 100-111.

[3] Aversa, R., Di Martino, B., Rak, M., and Venticinque, S. 2010. "Cloud agency: A mobile agent based cloud system." Complex, Intelligent and Software Intensive Systems (CISIS), IEEE International Conference on Complex, Intelligent and Software Intensive Systems. 132-137.

[4] Barella, A., Carlos C., and Botti, V. 2007. "Agent architectures for intelligent virtual environments." Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. IEEE Computer Society. 532-535.

[5] Cao, J., Spooner, D. P., Jarvis, S. A., and Nudd, G. R. 2005. "Grid load balancing using intelligent agents." Future generation computer systems 21.1: 135-149.

[6] Durupinar, F. 2010. From audiences to mobs: Crowd simulation with psychological factors. Ph.D. Dissertation. Bilkent University. Ankara, Turkey. July.

[7] Grimaldo, F., Lozano, M., Barber, F., and Vigueras, G. 2004. "Simulating socially intelligent agents in semantic virtual environments." Knowledge Engineering Review 23.4: 369-388.

[8] Herrero, P., Bosque, J. L., and Perez, M. S. 2007. "An agents-based cooperative awareness model to cover load balancing delivery in grid environments." On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops. Springer, Berlin Heidelberg. 64-74.

[9] Huhns, M. N. 1987. Distributed artificial intelligence. Morgan Kaufmann.

[10] Luck, M. and Aylett, R. 2000. "Applying artificial intelligence to virtual reality: Intelligent virtual environments." Applied Artificial Intelligence 14.1: 3-32.

[11] Macedonia, M. R. 1995. A Network Software Architecture for Large Scale Virtual Environments. Ph.D. Dissertation. Naval Postgraduate School. Monterey, California. June.

[12] Maher, M.-L. and Gero, J. S. 2002. "Agent Models of 3D Virtual Worlds." ACADIA 2002: Thresholds. California State Polytechnic University, Pamona, California. 127-138.

[13] Mendez, G., Perez, P., and de Antonio, A. 2001. "An overview of the use of mobile agents in virtual environments." In Proceedings of the 3rd International Workshop on Intelligent Virtual Agents (IVA). Lecture Notes in Artificial Intelligence, vol. 2190. Springer-Verlag, 126-136.

[14] Satoh, Ichiro. 2010. "Mobile Agents." Handbook of Ambient Intelligence and Smart Environments. Springer Science+Business Media, United States. 771-791.

[15] Thalmann, Daniel. 2001. "The foundations to build a virtual human society." In Proceedings of the 3rd International Workshop on Intelligent Virtual Agents (IVA). Lecture Notes in Artificial Intelligence, vol. 2190. Springer-Verlag, 1–14.

[16] Ullrich, S., Bruegmann, K., Prendiger, H., and Ishizuka, M. 2008. "Extending mpml3d to second life." In Proceedings of the 8th International Workshop on Intelligent Virtual Agents (IVA). Lecture Notes in Artificial Intelligence, vol. 5208. Springer-Verlag, 281–288.

[17] Weiss, G. (Ed.). 1999. Multiagent systems: a modern approach to distributed artificial intelligence. The MIT press.