CoABS Grid Scalability Experiments

Martha L. Kahn
Global InfoTek, Inc
mkahn@globalinfotek.com

Cynthia Della Torre Cicalese
Marymount University and Global InfoTek, Inc.
cindy@globalinfotek.com

ABSTRACT
The CoABS Grid is middleware that integrates heterogeneous agent-based systems, object-based applications, and legacy systems. It includes a method-based application programming interface to register agents, advertise their capabilities, discover agents based on their capabilities, and send messages between agents. All agent-to-agent communication is point-to-point. For this reason, it scales to a large number of agents with no restrictions beyond those imposed by network bandwidth. Agent registration and discovery, on the other hand, are reliant on one or more lookup services. The experiments discussed in this paper investigate how timing for agent lookup varies as the number of agents increases. No degradation in lookup performance was observed in experiments with up to 10,000 agents. Further experiments are planned.

Keywords
agent, CoABS Grid, Jini™, scalability

1. INTRODUCTION
In this paper we will give a brief introduction to the Control of Agent-Based Systems (CoABS) research program and the CoABS Grid framework for integrating agent-based systems [1]. The CoABS Grid implementation will be described in order to give context to the CoABS Grid scalability experiments that are the main focus of this paper. The CoABS Grid is based on the Jini™ Connection Technology. Therefore, the CoABS Grid scalability experiments can also be seen as Jini™ scalability experiments. Jini™ will also be discussed as a background to the experiments.

In these experiments, we registered 500 agents at a time with the CoABS Grid, until a total of 10,000 agents were registered. We performed twenty-two different agent queries after each group of 500 agents was registered, measuring how long it took to fetch the agents and the number of agents retrieved. In this paper, we present the design, implementation, and results of the experiments in detail, followed by our future experiment plans and conclusions.

2. BACKGROUND
2.1 CoABS

CoABS is a research program of the US Defense Advanced Research Projects Agency and the US Air Force Rome Labs. The aim of the program is to investigate the use of agent technology to improve military command, control, communication, and intelligence gathering. Over twenty universities and companies are participating in the research effort. Each participating organization brings to the program its own agent architecture, with different agent communication languages, ontologies, and agent-based services.[1] For instance, the CoABS Grid is being used to integrate RETSINA agents from Carnegie Mellon University [2], TEAMCORE agents from the University of Southern California Information Science Institute [3], and AAA agents from the Oregon Graduate Institute [4], to name just a few. Mobile agent systems being integrated include D'Agents from Dartmouth College [5], EMAA from Lockheed Martin, and Nomads from the University of West Florida [6].

The military environment provides many challenges that are addressed by the CoABS program. The environment is very dynamic, with operations changing quickly, hardware and software moving, connecting, and disconnecting, and network bandwidth availability varying greatly. There are existing inflexible, but very important, stove-piped legacy systems that need to be integrated. There is information overload, with vastly increased data available, but inadequate tools to filter the data. Finally, the military environment is heterogeneous with multiple standards and interfaces, as well as multiple hardware and software platforms.

The CoABS Grid is a framework for federating heterogeneous agent systems. It is designed to meet the challenges of the military environment, as well as address the heterogeneity among the participating agent research communities. Although the CoABS Grid is being developed with a military application in mind, it is a general-purpose agent framework with potential use by a wide variety of applications.

The CoABS Grid enables the dynamic interoperability of distributed agents, objects, devices, and legacy systems. It supports dynamic registration and discovery of relevant participants and flexible run-time communications. The CoABS Grid is adaptive and robust. Systems can be added and upgraded without reconfiguring the network. Failed or unavailable components are automatically purged from the registry.

The CoABS Grid, as referred to in this paper, is only one part of the overall CoABS program – the plumbing that connects the components developed by all the CoABS researchers to solve real-world problems. The scalability of this infrastructure is

[1] More information on the program is available from the website http://coabs.globalinfotek.com. The CoABS Grid software may be obtained from this site as well.

what is addressed here. As the CoABS Grid has been used in more realistic military experiments, such as US Navy Fleet Battle Experiments and the Coalition Agents Experiment involving US and British participants, it has become important to investigate the scalability of the CoABS Grid infrastructure.

2.2 CoABS GRID IMPLEMENTATION

The CoABS Grid is built using the Jini™ Connection technology developed by Sun Microsystems. The robust and dynamic nature of the CoABS Grid is derived from Jini™. The CoABS Grid software is written in Java and also uses Java Remote Method Invocation (RMI) for inter-agent communication. Members of the CoABS research community have created proxies that integrate the CoABS Grid with agent systems written in C++ and Lisp and for the Palm Pilot KVM. The CoABS Grid takes advantage of three important components of Jini™:

1. the Jini™ concept of a service, which is used to represent an agent,
2. the Jini™ Lookup Service (LUS), which is used to register and discover agents and other services, and
3. Jini™ Entries, which are used to advertise an agent's capabilities.

A Jini™ service is a Java object that is serialized and stored in the LUS. The LUS supports lookup of services based on type, attribute values, and unique identifier. When a Jini™ client performs a lookup through the LUS, the service object is returned to the client. The service may optionally be a proxy that uses a remote connection to communicate back to the true service at a different location. The remote connection is transparent to the client and can be of any type, e.g. RMI, CORBA, or secure socket.

The LUS grants leases to registered services, assigns globally unique identifiers to services, and supports lookup of services. It is the service's responsibility to maintain its lease with the LUS, however Jini™ provides helper classes to do this automatically. If a service cannot maintain its lease because of either failure of the service or failure of the network connection between the service and the LUS, the service will be purged from the LUS, so that the LUS contents remain current.

Jini™ provides helper classes that use a multicast protocol to find any LUSs that are running within a local area network. No prior knowledge of the machine name or port that the LUS is running on is required. Jini™ provides a unicast protocol to find LUSs outside the local area network. Service registration is maintained in all local and distant LUSs. The registration is automatically propagated to any new LUS processes that are started. Multiple LUSs can be run for robustness and scalability. If one goes down, the others will still maintain registration and lookup. A sample LUS is provided in the Jini™ Development Kit and is currently used by the CoABS Grid. The version of the CoABS lookup methods used for these experiments perform a filtering of Jini™ lookup results, removing proxies to any agents that are not reachable. Test results from a new version, which has this filtering step removed, are currently being analyzed.

Jini™ services are described in the form of a Jini™ Entry. An Entry is a collection of service attributes that is stored in the LUS along with the service. Many Entries can be stored for a single service. An Entry is an object that has public fields, cannot contain primitive types, and has a no-parameter constructor. Any Serializable object that meets these criteria can be an Entry as long as it implements the marker Jini™ Entry interface. Entry templates are used in Jini™ and CoABS Grid lookup methods to match registered services. A null Entry field is a wildcard. Non-null fields are used for exact matching. Entry templates and service types can be used to filter the number of services that are downloaded from a LUS over the network. Predicates can than be used local to the client agent to further restrict the number of services returned.

The CoABS Grid uses Jini™ Entries for agent capability advertisements. The CoABS Grid provides several classes that implement the Entry interface, and CoABS Grid users are encouraged to add new ones that are relevant to their applications. The CoABSAgentDescription Entry has fields for agent name, description, organization, architecture, ontologies, content languages, display icon URL, documentation URL, and unique ID. The CoABSAgentDescription was used in the CoABS Grid experiments, as was a test Entry described later.

The CoABS Grid provides helper utility classes that are local to an agent and that hide the complexity of Jini™. These classes automatically find any LUS in both the local area network and user-designated distant machines. The CoABS Grid supports agent and service discovery based on Jini™ Entries and arbitrary predicates as well as by service type. The CoABS Grid also provides event notification when agents register, deregister, or change their advertised attributes.

The CoABS Grid defines a Jini™ service interface called the AgentRep, which is a proxy to the agent. This interface defines a method called addMessage(), which uses a remote connection to deliver a message back to the agent. Thus, when a client agent calls a CoABS Grid lookup method, a proxy that allows immediate direct communication back to the agent is returned. The client agent can include its own AgentRep in the message it delivers, so that two-way communication can be established with no further lookup. The CoABS Grid is transport neutral in terms of agent communication. The CoABS Grid defines the interface, but the agent proxy is free to use any transport in its implementation.

The CoABS Grid currently provides an AgentRep implementation that uses RMI for message transport. Other transport mechanisms are in development. An AgentRep downloaded to a client is connected to a MessageQueue object local to the agent using RMI. A MessageListener interface is also defined to allow agents automatic notification of incoming messages. Several classes of CoABS Grid messages are provided. Some include text messages only, while others allow data attachments. The CoABS Grid is language neutral; any agent communication language can be used. It is up to the communicating agents to decipher the contents of a message. The CoABS Grid also provides methods to send a message to a group of agents matching a particular template or satisfying a particular predicate.

Agent communication is fully distributed, in that each agent sending a message communicates directly with the receiver, using the proxy registered by the receiver. The sender is unaware of the transport mechanism being used, although currently RMI is the default. Thus, as the number of agents on the CoABS Grid increases, agent communication performance is only affected by the distribution of the agents in the network, the network bandwidth, and the details of the particular AgentRep implementation. For this reason, this portion of the architecture is thought to be highly scalable, and no experiments on scalability of agent communication are currently planned.

3. EXPERIMENT DESIGN

The experiments presented here were designed to test the scalability of the CoABS Grid with respect to agent lookup. Since, the CoABS Grid uses the LUS for agent registration and lookup, the experiments also test the scalability of the LUS. They test lookup using Jini™ Entries. Experiments based on lookup using service type are planned for a later time. Filtering based on predicates is done local to an agent rather than by the LUS, so is not the focus of any planned experiments.

The experiments were not designed to give a definitive quantitative measure to the scalability of registration and lookup of the CoABS Grid. Instead, they were designed to give a qualitative understanding of whether performance problems become apparent with a highly populated LUS. As such, while the experiments were performed under conditions of light network load, no effort was made to completely isolate the network for the purposes of the experiments. Also, no effort was made to create a testbed of identical machines on which to run the various components of the experiments. The timing numbers presented in the next section of this paper are to be regarded as indications of performance trends, not as definitive indications of the time it takes to perform various functions. These experiments focus on how performance of agent lookup is affected as the LUS becomes more populated.

In order to be able to perform these tests, a test agent class was created. A test agent contains a reference to an instance of class TestEntry, which implements Entry. An agent's TestEntry is provided to the LUS when the test agent registers and is used to look up the agent. The attributes of TestEntry are shown in Table 1. The attributes are used to allow lookup experiments that return varying numbers of test agents based on the attribute values requested. In this fashion, individual lookup experiments were designed to return zero, one, or multiple matching test agents.

An agent factory was implemented to create and register test agents, providing them with an identifying TestEntry instance. The agent factory receives on the command line two parameters: *range*, the number of agents to be created and registered, and *offset*, the starting number to use for the sequenceNumber attribute. For example, if a range of 500 and an offset of 1000 are passed in to the agent factory, 500 agents will be created with sequence numbers of 1000, 1001, 1002, etc.

In addition, the agent factory creates a pseudorandom permutation of the integers between *offset* and *offset + range - 1*. The numbers in the permutation are used to assign the *size* and *name* attributes to each agent's TestEntry. So, if *range* is 500, *offset* is 1000, and the first number generated in the permutation is 1249, the first agent created, with a sequence number of 1000, will have *size* set to 1249 and *name* set to "agent-1249". This was done to allow the lookup experiment to be more independent of the order in which agents were registered.

The agent factory assigns the *color* and *language* attributes to the agents' TestEntries sequentially. That is, the first agent created will be assigned an entry with a *color* of Color.black and a *language* of "English". The second agent created will be assigned an entry with a *color* of Color.blue and a *language* of "French", and so on. Since the number of colors is not the same as the number of languages, the pairings of *color* and *language* will vary.

3.1 Looking Up Agents

All experimental machines were Pentium II or Pentium III class machines running Windows NT or Windows 2000. The machines were located on two subnets within one building with multicasting enabled between the subnets. One machine was designated to run the LUS. Twenty other machines were designated to run 500 agents apiece. A twenty-third machine was designated to run the actual lookup experiments. The agent factory on the first machine was given an *offset* of 0 and a *range* of 500. The agent factory on the second machine was given an *offset* of 500 and a *range* of 500. The agent factory on the third machine was given an *offset* of 1500 and a *range* of 500, and so on. The number 500 was chosen because it was a large number of agents that could reliably be run on the chosen agent machines without overloading them. After each new agent factory completed its registration of 500 agents, a lookup experiment was run. The lookup experiment involved making the following 22 lookup requests to the LUS:

1. return all agents with size = 0
2. return all agents with size = 500

. . .

19. return all agents with size = 9,500
20. return all agents with size = 10,000
21. return all agents with color = Color.blue
22. return all agents with color = Color.blue and language = "French"

Table 1. TestEntry Attributes

Attribute Name	Attribute Class	Attribute Description
name	String	a string of the form "agent-x" where x is an integer and is identical to the size attribute
color	java.awt.Color	one of: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow
language	String	one of: "English", "French", "Italian", "Spanish", "Dutch", "German", "Hungarian"
size	Integer	a pseudo-random integer
sequenceNumber	Integer	an integer indicating the order in which the agent was created (e.g. first agent or forty-third agent)

These lookups were done using a TestEntry template with exact matching of the relevant TestEntry *size*, *color*, and *language* fields, with all other fields left null to indicate wildcards. The first 20 lookup requests return either zero or one agents, depending upon whether an agent of the requested size has already been registered with the LUS. That is, after the first agent factory completes its agent registration, an agent of *size* equal to zero will be located, but the other size-related lookups will return zero agents. After the second agent factory completes its agent registration, agents of *size* equal to zero and 500 will be located, but the other size-related lookups will return zero agents, and so on. Since the size attribute is generated from a pseudorandom permutation of the integers between 0 and *offset + range – 1*, the agent with *size* equal to zero will not necessarily be the first agent registered with the LUS.

The last two lookup requests each return multiple agents in a single response. The first looks for all agents with *color* equal to blue, and the second for all agents with *color* equal to blue and *language* equal to French. In each case, as more agents are registered, more agents are returned from the lookup.

The application that requests the lookup records the length of time from when the lookup is requested to when the response is received. Since the timing is done on the lookup machine, not in the LUS, it includes the network delay in transmitting the lookup requests and results.

4. EXPERIMENT RESULTS

Table 2 presents a summary of the results of the lookup experiments. Each row of the table holds data for one of the machines used in the experiment. The first two rows show the processor information for the machines running the LUS and the lookup tests. The rest of the rows show data for the machines running agent factories, presented in the order in which the agent factories were started, from the top of the table down. Each row of lookup timing information was filled in with data from the lookup test performed following running the agent factory for that row. In the columns that report the least time to

Table 2. Lookup Experiment Results Summary

Processor	# Agents (On This Machine)	# Agents (Cumulative)	Least Time to Look Up 1 Agent* (ms.)	Least Time to Look Up 0 Agents* (ms.)	Time to Look Up Blue Agents** (ms.)	Number Blue Agents Found**	Time to Look Up Blue/French Agents*** (ms.)	Number Blue/French Agents Found***
PIII 733 MHz	Jini™ LUS	N/A	N/A	N/A	N/A	N/A	N/A	N/A
PIII 600 MHz	Lookup Tests	N/A	N/A	N/A	N/A	N/A	N/A	N/A
PIII 400 MHz	500	500	90	10	771	39	200	6
PII 500 MHz	500	1,000	50	10	1543	77	310	11
PIII 400 MHz	500	1,500	80	10	1863	116	471	17
PIII 733 MHz	500	2,000	70	10	2664	154	511	22
PIII 733 MHz	500	2,500	80	10	2774	193	661	28
PIII 500 MHz	500	3,000	70	10	3104	231	801	33
PII 300 MHz	500	3,500	70	10	4446	270	831	39
PIII 866 MHz	500	4,000	70	10	4417	308	1002	44
PII 450 MHz	500	4,500	80	10	4947	347	1171	50
PIII 450 MHz	500	5,000	70	10	4917	385	1082	55
PII 400 MHz	500	5,500	70	10	5828	422(424)	1241	61
PII 450 MHz	500	6,000	70	10	5848	462	1302	66
PII 233 MHz	500	6,500	70	10	7361	498(501)	1502	72
PIII 733 MHz	500	7,000	70	10	7191	539	1402	77
PIII 650 MHz	500	7,500	70	10	8142	577(578)	1662	83
PIII 550 MHz	500	8,000	70	10	7681	613(616)	1602	87(88)
PIII 400 MHz	500	8,500	70	10	9393	653(655)	1943	93(94)
PIII 400 MHz	500	9,000	70	10	11377	692(693)	5047	98(99)
PIII 677 MHz	500	9,500	70	10	9604	730(732)	2113	104(105)
PIII 733 MHz	500	10,000	70	N/A	9884	769(770)	2123	109(110)

* Twenty separate lookup tests: looked up agents with size = 0, 500, 1000, 1500, …..10,000 – zero or one agent of each size

** One attribute lookup test: find all agents with TestEntry.color = Color.blue

*** Two attribute lookup test: find all agents with TestEntry.color = Color.blue and TestEntry.language = "French" look up one and zero agents, the time reported is the minimum time for the 20 size lookup requests returning one and zero agents, respectively. The least time was chosen since the experimental results, in general, are quite consistent, aside from several spikes that are likely due to network latency, garbage collection, and generating a LUS snapshot file. The results of the 20 individual size lookup requests for each experiment are presented in Table 3.

The following figures present a graphical depiction of the various experiment results. These graphs help to highlight the fact that the LUS performance was not found to degrade for sequential lookup when populated with up to 10,000 agents.

Figures 1 and 2 present graphs showing that LUS performance for looking up zero or one agent does not degrade with increased population. Figures 3 and 4 present graphs showing that lookup performance of the LUS appeared to degrade when looking up multiple agents as the LUS was populated with up to 10,000 agents. The number of agents returned from the lookup requests, however, also increased as the population of agents increased, since there were more matches in the LUS.

Therefore, we normalized the results by the number of agents found. Figures 5 and 6 show the normalized graphs, which indicate that lookup of multiple agents is roughly proportional to the number of agents retrieved.

We hypothesize that this increased time to look up multiple agents is due in part to the Grid's filtering out of non-reachable agents. In order to determine if an agent is reachable a remote method invocation is made to each agent returned from the LUS. Thus, there is an RMI call made for every returned agent. We have conducted experiments with the latest version of the Grid with this filtering step removed and are in the process of analyzing the results. Other variables that may account for the slope of these graphs are increased LUS processing time and increased network delay to transmit the greater number of responses.

We also believe that the filtering out of unreachable agents accounts for some missing data in Table 2. The cells of Table 2 that show two numbers indicate where discrepancies were observed between the number of agents returned from a lookup, and the number that were expected. The numbers in parenthesis indicate the expected number of agents. We hypothesize that the missing agents in these runs were due to either transient network problems or agent threads that died on local machines.

Table 3. Size Lookup Experiment Results

Number of Registered Agents

Size Looked Up	**500**	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**	**5000**	**5500**	**6000**	**6500**	**7000**	**7500**	**8000**	**8500**	**9000**	**9500**	**10000**
0	**90**	**60**	**110**	**70**	**80**	**70**	**80**	**90**	**110**	**80**	**70**	**70**	**80**	**80**	**70**	**80**	**70**	**90**	**70**	**110**
500	*10*	**50**	**81**	**71**	**80**	**80**	**91**	**91**	**80**	**80**	**80**	**80**	**70**	**80**	**80**	**80**	**80**	**100**	**80**	**70**
1000	*11*	*10*	**80**	**80**	**90**	**80**	**90**	**80**	**90**	**70**	**80**	**70**	**80**	**100**	**81**	**81**	**70**	**80**	**80**	**90**
1500	*10*	*10*	*20*	**80**	**80**	**90**	**70**	**80**	**80**	**80**	**70**	**70**	**70**	**70**	**70**	**80**	**80**	**70**	**80**	**70**
2000	*10*	*10*	*21*	*11*	**381**	**70**	**71**	**70**	**80**	**70**	**90**	**80**	**80**	**80**	**70**	**80**	**80**	**80**	**71**	**70**
2500	*10*	*10*	*10*	*10*	*10*	**70**	**90**	**80**	**81**	**70**	**90**	**80**	**80**	**71**	**70**	**80**	**70**	**70**	**120**	**80**
3000	*10*	*10*	*10*	*10*	*10*	*10*	**1252**	**441**	**90**	**100**	**90**	**90**	**80**	**521**	**90**	**130**	**80**	**90**	**100**	**80**
3500	*10*	*10*	*10*	*20*	*10*	*10*	*10*	**70**	**80**	**80**	**70**	**81**	**81**	**90**	**80**	**80**	**70**	**70**	**80**	**80**
4000	*10*	*10*	*10*	*10*	*10*	*10*	*11*	*10*	**81**	**80**	**70**	**80**	**90**	**80**	**80**	**70**	**80**	**80**	**80**	**70**
4500	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*20*	**110**	**70**	**130**	**80**	**80**	**80**	**80**	**80**	**70**	**80**	**80**
5000	*10*	*20*	*10*	*10*	*20*	*10*	*10*	*10*	*10*	*10*	**80**	**90**	**91**	**90**	**90**	**90**	**90**	**90**	**90**	**540**
5500	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*20*	*10*	*10*	**90**	**90**	**80**	**90**	**80**	**90**	**80**	**120**	**80**
6000	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	**110**	**271**	**110**	**180**	**110**	**110**	**110**	**110**
6500	*20*	*10*	*10*	*10*	*10*	*11*	*10*	*10*	*10*	*10*	*10*	*20*	*10*	**130**	**90**	**100**	**130**	**330**	**160**	**80**
7000	*10*	*10*	*10*	*20*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	**70**	**70**	**80**	**80**	**80**	**80**
7500	*10*	*20*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*250*	*10*	*10*	*20*	*10*	**90**	**80**	**80**	**80**	**80**
8000	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*20*	*10*	*10*	*10*	*10*	*11*	*10*	*10*	**91**	**90**	**90**	**90**
8500	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*20*	*10*	*30*	*10*	*10*	*10*	*10*	**100**	**90**	**90**
9000	*10*	*10*	*10*	*10*	*20*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*110*	*20*	*10*	**80**	**71**
9500	*10*	*20*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*20*	*10*	*10*	*10*	*20*	*30*	*20*	*10*	*10*	**80**

Bold: lookup returned one agent

Italic: lookup returned zero agents

Figure 1. Lookup Experiment Results – Looking Up One Agent

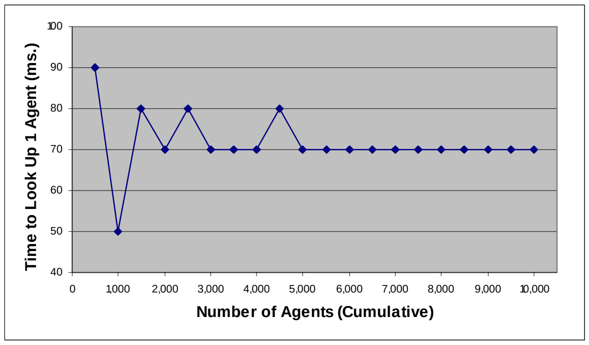


Figure 2. Lookup Experiment Results – Looking Up Zero Agents

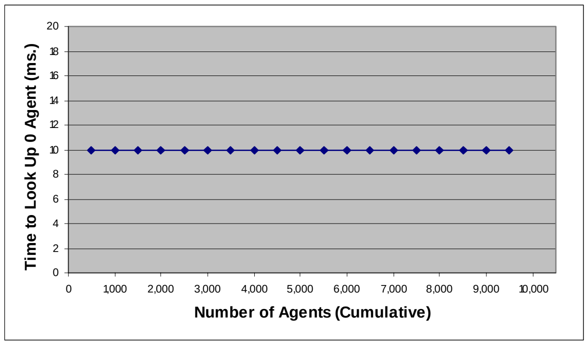


Figure 3. Lookup Experiment Results – Looking Up Multiple (Blue) Agents

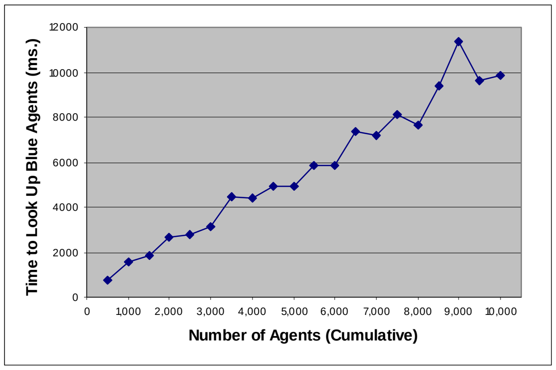


Figure 4. Lookup Experiment Results – Looking Up Multiple (Blue and French) Agents

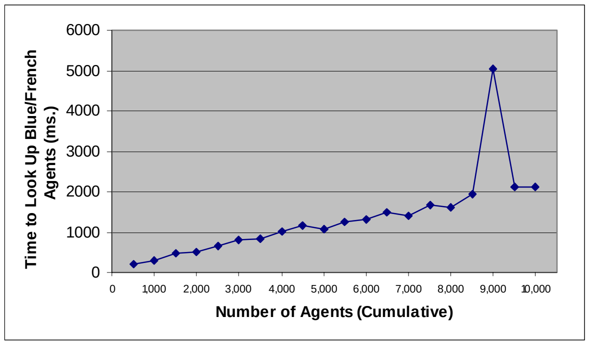


Figure 5. Normalized Lookup Experiment Results – Looking Up Multiple (Blue) Agents

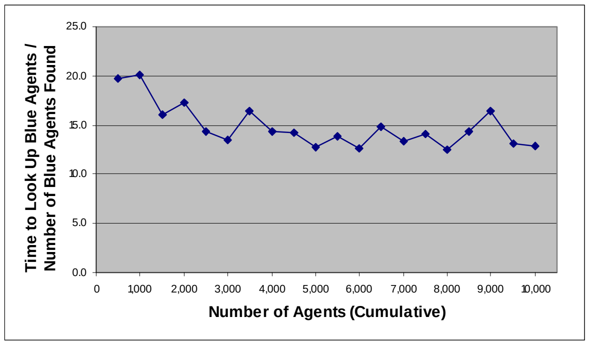
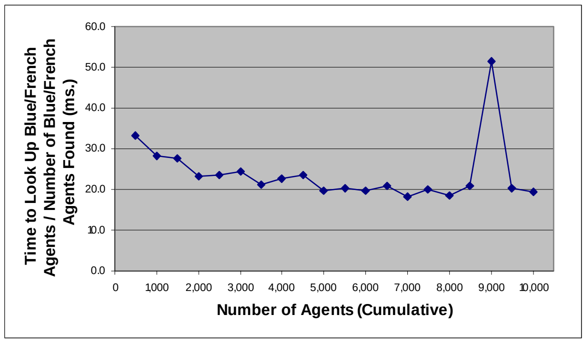


Figure 6. Normalized Lookup Experiment Results – Looking Up Multiple (Blue and French) Agents



5. ONGOING AND FUTURE WORK

We are currently conducting experiments with registration time measured as the LUS becomes more populated. We wish to measure both sequential and concurrent registration. We also have rerun the experiments with a newer version of the Grid with the reachability filter removed, as already discussed.

There are several more CoABS Grid scalability experiments that we would like to perform. We would like to repeat our experiments with more than one LUS. We would like to perform experiments that look up agents based on service type, in addition to Entry templates. To do this, we will extend the AgentRep with subclasses that implement several different interfaces. We can then test lookup based on a combination of class, superclass, or interface type. We also would like to perform experiments with more than one Entry Template (lookups would return only services that match all Entries). Experiments with combinations of service type and Entry templates would also be useful as would parallel lookup experiments.

We would like to extend the experiments past 10,000 registered agents to determine at what point the performance of the LUS begins to degrade. Members of the CoABS research community from universities across the country have volunteered to help us with further experiments.

Finally, it would be interesting to perform the same set of experiments on a commercial LUS to see how they compare.

6. CONCLUSIONS

This experiment has proven that the scalability of the CoABS Grid, with respect to agent lookup, is excellent up to at least 10,000 registered agents when lookup is sequential. Lookup of agents with zero and one matches is not affected by number of agents registered. Time for the lookup of many agents increases approximately proportionally to the number of agents looked up. Finally, time to look up multiple agents appears to be independent of the number of agents registered.

7. ACKNOWLEDGMENTS

The CoABS Grid was funded by DARPA under its Control of Agent-Based Systems program through Air Force Research Laboratory Contract No. F30602-98-C00267. We would like to thank our sponsors Jim Hendler of DARPA and Dan Daskiewich of AFRL for supporting our research. We would like to thank Darrin Taylor and Lee Alford of 21st Century Technologies who performed analyses of the experimental results. We would also like to thank the other members of the CoABS Grid development team. Team members from Global InfoTek, Inc. include Andrew Perez-Lopez, who assisted with the experiments, Dennis Brake, Bernadette Clemente, Arundhati Jayarao, Phil Sage, and Doyle Weishar. Team members from ISX Corp. include Brian Kettler and Sandra Powers.

8. REFERENCES

[1] Hendler, J. and Metzeger, R., “Putting it all together – The Control of Agent-Based Systems and Their Applications,” IEEE Intelligent Systems, vol. 14, No. 2, Mar. 1999 p.37.

[2] Giampapa, J.A., Paolucci, M., and Sycara, K., Agent Interoperation Across Multagent System Boundaries, Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000), Association for Computing Machinery, ACM, 1515 Broadway, New York, NY, 10036, USA, June, 2000.

[3] Pynadath, D., Tambe, M., Arens, Y., Chalupsky, H., et al Electric Elves: Immersing an agent organization in a human organization. Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents --- the human in the loop (from http://www.isi.edu/teamcore/tambe/)

[4] Kumar, S. & Cohen, P. "Towards a Fault-Tolerant Multi-Agent System Architecture." In Proceedings of The Fourth International Conference on Autonomous Agents (Agents 2000), Barcelona, Spain, June 3-7, 2000, ACM Press, pp. 459-466.

[5] Gray, R. “Agent Mobility: Performance, Security and a Case Study”, Tutorial presented at PAAM 2000, Manchester, England, April 10-12, 2000

[6] Suri, N., Bradshaw, J., Breedy, M., Groth, P., Hill, G., Jeffers, R., Mitrovich, T., Pouliot, B. and Smith, D. "NOMADS: toward a strong and safe mobile agent system" In Proceedings of The Fourth International Conference on Autonomous Agents (Agents 2000), Barcelona, Spain, June 3-7, 2000, ACM Press, pp. 163-164.