A Performance Evaluation of the ACORN Architecture

Virendra C. Bhavsar, Ali Akabar Ghorbani
Faculty of Computer Science, University of New Brunswick Fredericton, NB.
{bhavsar | ghorbani}@unb.ca

Stephen Marsh
Interactive Information Group, Institute for Information Technology, NRC, Ottawa, ON.
{steve.marsh}@iit.nrc.ca

ACORN (Agent-based Community Oriented Retrieval Network) is a multi-agent system which uses agents to provide information across internet/intranet networks. In this report, we adapt the ACORN architecture for its performance evaluation on single and multiple servers, running on single and multiple machines. In order to evaluate the performance of ACORN, we introduce a novel concept of multiple autonomous virtual users. The concept of multiple autonomous virtual users and our testing philosophy is applicable to the performance evaluation of other client/server based multi-agent systems. The modified ACORN architecture has been ported to different machines and experimental results on single processors obtained. The processing time required by ACORN is found to be a nonlinear function of the number of agents.

1. Introduction

The explosive growth of the online information makes it increasingly difficult for users to locate and exploit the information available worldwide. Autonomous agents allow a radically new approach that makes information access easy and efficient. Rather than the user having to go and track down information (the 'pull' model), the agents provide a unique ability to profile, target and help their customers. Information filtering algorithms, for example, automated collaborative filtering used in the Firefly products [2] and the artificial neural networks (ANNs) used in the Autonomy products [1], are central to the development of intelligent agents that search the very large number of databases and compare the profiles of millions of users.

In order to satisfy the real-time response requirements of complex learning algorithms and exploit the capabilities of high performance multiprocessor web servers, we carried out a performance evaluation of the ACORN Multi-Agent Information [8, 9, 10, 11]. ACORN is a system developed at the Interactive Information Group, NRC, Ottawa. ACORN uses agents to provide active information across networks. ACORN promises to

be a very useful technology for many companies involved in the development of information search and dissemination products. However, there are many issues that need to be studied further before ACORN can be integrated into commercial products: testing with a large number of users, testing over geographically distributed networks, and scalability of ACORN on multiprocessor web servers. Testing of ACORN with large numbers of human users is an almost insurmountable task. In order to tackle this problem we have taken a novel approach which uses multiple autonomous virtual users.

This report describes the work carried out as a part of a c3.ca pioneer project. The next section describes the ACORN architecture in brief, including related work. Section 3 discusses the concept of multiple autonomous virtual users, and section 4 provides a discussion of the changes made to ACORN for the purpose of allowing the creation and use of multiple autonomous virtual users to facilitate the performance evaluation of ACORN on single and multiple (sequential and parallel) servers. Section 5 presents details of our experimental design and some initial results. Conclusions and ideas for future work are given in Section 5.

2. ACORN

ACORN is a multi-agent architecture using autonomous agents to do the work of disseminating and searching for information in internet/intranet networks. While ACORN has been discussed and described more fully elsewhere [8, 9, 10, 11], we will briefly discuss it here, along with related work.

2.1 Background

Information is hard to work with, particularly if there's a lot of it and it's disorganized, such as on the Web. Current solutions are straining at the seams, and a new paradigm is required to handle the volume and noise. Agents present only a possible solution, but one which bears closer investigation. Conceptually, an agent can be sent out onto networks to scour web sites or corporate information sites and databases available to it for information relevant to a search topic it has been given. This is an extension to the 'spiders' original search indexes used, with directions not to grab every file at a site, but to get relevant pages. Such 'searchbots' exist and work relatively well, for all their simpilicity. However, humans employ other search strategies, via their communities: asking questions of people you know might result in them going elsewhere to ask another person they know (and so on) resulting in, eventually, an answer coming back to them. This is community-based searching, and requires only that you are a member of a

community, and since communities overlap, it's easy to grab information across community boundaries.

2.2 Related Work

Community-based navigation and information sharing is not new [4, 6, 7] and interesting twists on the theme exist, such as Firefly [2], which take the views of a community, aggregate them, match individual interests, and recommend to their members movies, music or whatever that related members are into. These services work, but in a limited sense: people need to go to some effort to describe their interests, and they need to keep coming back to recommend new movies, recordings, books, or whatever. Amazon (www.amazon.com) has another way of doing this, by suggesting books to people that others have bought, based on the title they are looking at now. This requires no effort on the part of the 'recommender', and works surprisingly well. The concept of effort is important: people aren't going to use something they have to do extra work for unless they get something more in return.

Agents have been used to provide or discern links between people. Foner's *Yenta* [3] is an example of this. It takes people and their interests and tries to match them with others with similar interests. This is of particular use in a community-building situation, but can also help when looking for information (or possible interested receivers of information you've created); similar systems are described in [5, 6].

2.3 ACORN Architecture

ACORN takes community-based systems to a new level by using Milgram's Small World problem [12] and the concept of 'agent as information'. In ACORN, every piece of information - a document, part of a document to be distributed in isolation, an image, a sound, in fact, anything that can be distributed - is represented by an agent. Each agent has some 'knowledge' of its creator, its content, and importantly, its community, both of other agents, and human users. In this section, we will briefly describe how ACORN works and its current status and architecture.

ACORN is, from the top level, a Client/Server architecture. Clients, the user's interface to the ACORN system, provide the ability to create agents, organize received agents, describe a user's interests, and input details of other users and their interests. This information is used in the agents to better distribute themselves. Currently, interests are described in terms of simple keyphrases, for example, John is interested in Cars, Vegetarian Cookery, Agents, and Sugar Refining. If Mary produces a document about the

refining of sugar in Outer Mongolia, and decides to distribute this via ACORN, the Client will suggest she send the document to John (amongst others she may know). All that is required is that Mary inform the Client initially of John's interests. Moreover, when Mary's agent goes out to distribute its information, it may encounter other agents, say one belonging to Melissa, whose interests include Outer Mongolian politics and industry. Mary's agent will automatically note Melissa's interests and add her to its list of receivers (it'll get to her eventually). Since the agent has to return home on completion, Mary's Client will ultimately learn of Melissa's interests and add them to its database. Next time Mary produces a document about Outer Mongolia, Melissa's name will be suggested. If sugar isn't involved, John's won't. All Client 'knowledge' is stored in a separate file, called the 'client core'. This core contains user information, agents received and sent, user preferences, and also information about the people in the user's database (in our example, John and Melissa). Data about other users is stored as email addresses allied with keyphrases.

The Servers in ACORN provide migration/mobility capabilities to agents, and also give ACORN a degree of persistence in information. All Client data is stored (as a client core) on the server side as well as the client side, so that, for example, if the client shuts down, incoming agents can still be stored ready for downloading to the user's local machine. By the same token, we can provide automatic redirection and recommendations to agents because the server can access the client cores to match agent topics with known user interests in the client core.

Agent migration is achieved by sending agent cores from server to server. Like client cores, the agent core contains all the information the agent needs to accomplish its tasks: addresses of people to visit, user interests, document information and so on. When an agent reaches a new site, the server at that site instantiates a new agent with the core that has just been received, and lets the agent go on its way. At a site, the agent can visit users, and communicate with (mingle with) other agents at a central meeting point (which we call a *café*) to share addresses and interests with others, thus aiming to extend the list of addresses for people to visit.

3. Multiple Autonomous Virtual Users

Previously, ACORN had been tested only in a limited arena, with few users and only a few machines for servers. Obviously this was an inadequate situation for an architecture which is aimed at multiple users over a large network, where ultimately we are aiming for many thousands of users and servers in a stable and speedy system. Unfortunately,

distributing a system for testing to as many people as we need proves not only unwieldy but also unwise: it would be better if the potential users weren't put off by the system's initial problems. What is needed, then, is a way of testing ACORN without the involvement of a thousand or so willing (or otherwise!) human helpers.

This project aims to provide just such a testbed, using Multiple Autonomous Virtual Users to simulate a working network of people. In the system, we implemented several autonomous servers, each of which serves many autonomous virtual users. Each virtual user will be capable of creating agents to distribute and/or search for information available from others in the system. Each will 'know' something of at least one other user in the complete testbed: interests, address, and so on. Out aim is to set up the testbed so that we can observe several aspects of ACORN's working, including the processing time involved in the manipulation of multiple information agents at various stages in their execution, such as in their mingling in cafés. Importantly, we also examine the *scalability* of the architecture so that we can improve its workings.

A virtual user in our testbed is a process which creates agent cores, using data from individual client cores. These agent cores are instantiated at the server and execute in ACORN as if they were created by a real user. Subsequently, these agent cores may migrate to other servers. On arriving at a new server, they are forwarded as necessary to new virtual users. Thus on the receiving side the virtual user does little, as envisaged presently, other than provide a placeholder for a client core.

The creation of new agents is done as follows. A virtual user picks a topic from its client core's interest list at random and produces a new agent for this topic. It looks in the client core for details of relevant interested parties (virtual users), and adds these to the agent's destination list. An agent should have at least one potential destination user, although in a future version of ACORN, we intend to remove this restriction so that agents can find their own receivers from scratch. The agent is then sent out to do its work. This process is repeated at random intervals for each virtual user.

The receipt of an agent involves its being added to the client core by the server, and having relevant addresses added to its destination list. The client core should also be augmented depending on the agent's 'knowledge,' which we envisage the virtual user carrying out. The agent should possibly be delayed for a random interval before being sent out again in order to simulate a human reading the data, or being unavailable for a time, for example.

It is possible to imagine our virtual users performing many other tasks, from producing debugging messages to random 'failures' or moving to other sites, and so on, in order to simulate more closely human behaviours. However, we are interested primarily in scaling in the initial phase of testing, and envisage that additional tasks will become apparent as needed.

4. Adaptation of ACORN for Testbed

ACORN is a relatively large software system with more than 100 Java classes. Many of these classes are concerned with user interaction with ACORN via a windowing system on a client machine. In our testbed, we eliminate the need for a human user at the client machine by using autonomous virtual users. Thus, the need for a sophisticated user interface is removed. Additionally, since we are interested primarily in the execution of agents on the server side, we have no need for client behaviour classes. Our first task was thus to identify and remove all extraneous classes from the complete ACORN system.

ACORN in reality is a client-server architecture running over multiple servers. In order to simulate this situation, we found it convenient to be able to run ACORN with more than one server on a single machine. This also gives us the ability in future to make use of multiple processors on a single machine. To facilitate this, we needed to adapt the architecture with the addition of a SiteController class (see figure 1). The SiteController handles all communications between servers on a single machine, resolving any agent migration requests to different servers on that machine. The SiteController also handles communication between different machines. The way this works is that communications between machines are between SiteControllers on each machine, and each SiteController routes agents to the correct server on its own machine. Figure 1 shows this implementation of the architecture. The Streamer class provides transport of agents across IP.

An added bonus of our approach was the removal of the need for continuous user interaction with ACORN – our tests often required that the system run in batch mode due to the restrictions imposed on the parallel processing machines we were using (the c3.ca machines available via CA*Net were available to us only in batch mode). In addition, our adaptation removed much of ACORN's complexity, and resulted in a total of only 26 Java classes.

Referring to Figure 1, the reader will note the existence of an agent file and a log file at each site. The agent file contains a sequence of agent records, each of which is a textual

representation of an agent core. The SiteController reads these records sequentially and creates agent cores, sending them to their assigned Servers, at which point the system behaves exactly as original ACORN (see Section 2 and [8, 9, 10, 11]). The log file contains all system messages generated by ACORN. Most of these messages are used as debug messages and allow us to follow the execution of agents in the system, and will not be present in the final distribution version of ACORN.

5. Experimental Results

We ported the modified ACORN software to several machines: Sun Ultra Sparc1, 300 MHz Pentium II IBM PC, 100 MHz IBM Power PC, 300 MHz PowerPC G3 Mac, and an SGI ONYX L Series machine at the University of Windsor.

We selected the following parameters for our experiments. For each virtual user two keyphrases are used to specify user interests. Further, each agent contains four keyphrases about the document or search area. Finally, two recommended user addresses are included in an agent; for each recommended user, four keyphrases are used for to specify their interests.

Selected sample results are given in Table 1. In this table, the ACORN Execution Time refers to the time taken to start up, mingle all agents satisfactorily, and output test results. There are interesting points to note in these data. One can be noted in the seeming slow functioning of the SGI machine, contrary to expectations, and another in the extremely good performance of the PowerPC machine with small numbers of agents. We have no explanation for these, but are examining them more closely. What is noticeable about these results is a non-linear increase in processing time (despite the anomaly of the PowerPC). This may be due to the amount of mingling time required. The way the Café works results in $O(n^2)$ interactions between agents, where n is the number of agents. Moreover, an interaction between two agents involves a number of comparisons of keyphrases contained within each agent. Thus, the time taken is larger than expected with more agents because of several factors, some beyond our control (such as the overhead of java thread handling) and some within it (for example, the size – number of interests, number of recommended addresses, and so on - of each agent in the Café). These results give a foundation for building a performance model of ACORN with a view to refining and enhancing its performance.

Finally, although these results show that a large amount of processing time is needed for ACORN to handle relatively small numbers of agents, note that the execution times given

in Table 1 include significant amounts of screen I/O time. Further, the log file created contains large amounts of data, including detailed descriptions of all agent interactions. Initial results with little or no output in this respect (with debugging turned off) show a promising marked increase in speed.

6. Conclusion and Future Work

ACORN is a multi-agent system which uses agents to provide active information across internet/intranet networks. In this report, we have summarized our efforts in adapting the ACORN architecture for its performance evaluation on single and multiple servers, running on single and multiple machines. In order to evaluate the performance of ACORN, we have introduced a novel concept of multiple autonomous virtual users. The main objective of creating multiple autonomous virtual users has been accomplished. This philosophy is applicable to the performance evaluation of other client/server based multi-agent systems. We have ported the modified ACORN to different machines and obtained experimental results on single processors. The processing time required by ACORN is found to be a nonlinear function of the number of agents.

In order to support a large number of users, we envisage that a multiprocessor server would be necessary. In order to run the modified ACORN on multiprocessor machines, we need to carry out further changes to ACORN. Consequently, we think that ACORN should be further modified to adapt to multiprocessor machines, such as those available through C3.ca in Canada. Such a study is valuable in the context of the fact that very few systematic performance evaluation studies have been reported in the performance evaluation of multi-agent systems. Finally, the use of two or more multiprocessor machines running multiple virtual users at different sites would allow us to study more closely the performance of both ACORN and high performance networks in Canada.

Acknowledgements

The authors would like to acknowledge the pioneer project grant provided by C3.ca, Inc., in the partial support of this work. The first author also gratefully acknowledges the partial support from NSERC through Grant OGP0089. The support of the NRC in this project is also greatly appreciated.

References

[1] Autonomy Corporation, URL: http://www.agentware.com.

[2] Firefly Network, URL: http://www.firefly.net/

[3] Foner, L. & Crabtree, I. B. 1996. “Multi-agent matchmaking”, BT Technology Journal, Volume 14, number 4, pages 115-123.

[4] Hill,W., Stead, L., Rosenstein, M. and Furnas, G. "Recommending and Evaluating Choices in a Virtual Community of Use", in Proceedings CHI’95.

[5] Kautz, H., Milewski, A. and Selman, S., 1995, "Agent Amplified Communication", in Proceedings AAAI-95 Spring Symposium on Information gathering from Heterogeneous, Distributed Environments, Stanford, CA.

[6] Kuokka, D. & Harada, L. 1995. “Matchmaking for Information Agents”, in Proceedings International Joint Conference on Artificial Intelligence, Montreal, Canada. pages 672-678. Morgan Kaufmann, CA.

[7] Maltz, D. and Ehrlich, K. 1995. “Pointing the way: active collaboration filtering”, in Proceedings CHI’95.

[8] Marsh, S.P., 1997 “A Community of Autonomous Agents for the Search and Distribution of Information in Networks”, in (Ed.) Proceedings BCS-IRSG annual conference, Aberdeen, April.

[9] Marsh, S.P., 1997, ”A Different Approach to Information Provision and Retrieval”, in Frohmann, B. (Ed.) Proceedings CAIS’97, Annual Meeting of the Canadian Association for Information Science, St. Johns, Newfoundland, June.

[10] Marsh, S.P., 1997, ”Smart Documents, Mobile Queries: Information Provision and Retrieval using a Multi-Agent System”, in Ferguson, I. (Ed.), Proceedings IJCAI’97 Workshop on AI In Digital Libraries, Nagoya., Japan, August.

[11] Marsh, S.P. and Masrour, Y., 1997, Agent Augmented Community Information – The ACORN Architecture, in Proceedings CASCON97, Toronto.

[12] Milgram, S., 1992, “The Small World Problem”, in Milgram, S. (Sabini, J. and Silver, M. (Eds.)) “The Individual in a Social World: Essays and Experiments” , 2nd Edition. New York: McGraw Hill.

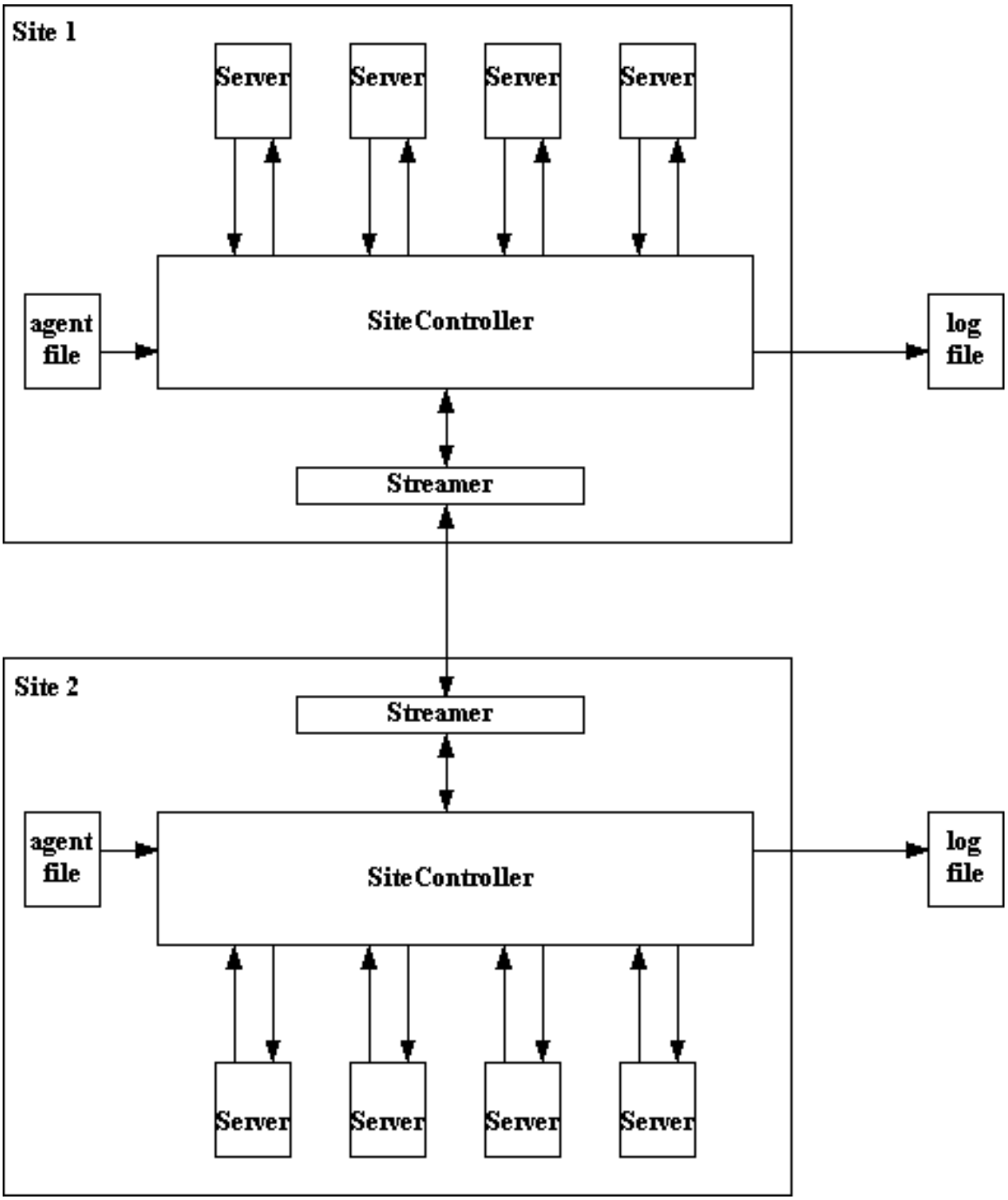

Figure 1. ACORN Testbed Structure (two machines).

Machine Type	Number of Agents	ACORN Execution Time (s)
300MHz Pentium II PC	20	50
	50	209
100MHz PowerPC (unix)	20	28
	40	41
	80	68
	160	888
	200	1052
SGI ONYX L[1]	20	32
	40	75
	80	240

Table 1. Sample Execution Results.

[1] We could not run the software for more than 80 agents. After about 10 minutes, the Internet connection to the SGI machine was automatically closed by the SGI host at the University of Windsor.