Mobile-Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task

Robert S. Gray[1], David Kotz[1], Ronald A. Peterson, Jr.[1], Joyce Barton[2], Daria Chacón[2], Peter Gerken[2], Martin Hofmann[2], Jeffrey Bradshaw[3], Maggie Breedy[3], Renia Jeffers[3], and Niranjan Suri[3]

[1] Dartmouth College Computer Science,[†] Hanover NH, USA,
`{rgray,dfk,rapjr}@cs.dartmouth.edu`
[2] Lockheed-Martin Advanced Technology Laboratory,[‡] Camden NJ, USA
`{jbarton,pgerken,mhofmann}@atl.lmco.com, dachacon@bigfoot.com`
[3] Institute for Human and Machine Cognition,[§] Univ. of West Florida (Pensacola) USA
`{jbradshaw,mbreedy,rjeffers,nsuri}@ai.uwf.edu`

Abstract. Building applications with mobile agents often reduces the bandwidth required for the application, and improves performance. The cost is increased server workload. There are, however, few studies of the scalability of mobile-agent systems. We present scalability experiments that compare four mobile-agent platforms with a traditional client/server approach. The four mobile-agent platforms have similar behavior, but their absolute performance varies with underlying implementation choices. Our experiments demonstrate the complex interaction between environmental, application, and system parameters.

1 Introduction

One of the most attractive applications for mobile agents is distributed information retrieval, particularly in mobile-computing scenarios. By moving the code to the data, a mobile agent can reduce the latency of individual steps, avoid network transmission of intermediate data, continue work even in the presence of network disconnections, and complete the overall task much faster than a traditional client/server solution.

A common performance concern about mobile-agent systems, however, is that they shift much of the processing load from the clients to the server. This shift is a significant advantage in some environments: the clients may be hand-held

computers with limited memory and computational power, and the "server" may be a large multiprocessor computer. On the other hand, the shift does raise questions about scalability. As the number of clients increases, how well do the mobile-agent services scale? Where is the trade-off between the savings in network transmission time and the possible extra time spent waiting for a clogged server CPU?

We set out to examine these questions. In the context of a simple information-retrieval application, we compared a traditional client/server (RPC) approach with a mobile-agent approach on four mobile-agent platforms. Our goal was to understand the performance effects that are fundamental to the mobile-agent idea, and separately, the performance effects due to implementation choices made by the different mobile-agent platforms.

We begin a comparison of the four mobile-agent systems we consider. Then we describe the scenario chosen for our tests, and the details of the tests themselves. We present the experimental results and our interpretation. Finally, we compare our results with the most relevant prior literature.

2   Mobile-Agent Systems

We evaluate four mobile-agent platforms: D'Agents [7, 8] from Dartmouth College, EMAA [12, 4] from Lockheed-Martin Advanced Technology Laboratory, KAoS [3, 2] from Boeing and the University of West Florida Institute for Human & Machine Cognition (UWF-IHMC), and NOMADS [14] from the UWF-IHMC. We chose these systems because they were available to us and because they represent a range of design choices, yet share a common language (Java). Since a full presentation of these systems is outside the scope of this paper, Table 1 outlines the features most relevant to our experiments. Each feature represents a design decision made by the systems' authors. We discuss the importance of these decisions here.

KAoS uses a hybrid approach allowing static agents to dispatch small task-specific agents called *minions*. KAoS allows developers to plug in different mobility solutions. In this case, our experiments used Voyager 3.0 for KAoS mobility, so most performance effects are dependent upon Voyager's implementation.

(a) D'Agents and NOMADS support strong mobility, where the agent's *control state*, as well as its code and data state, is moved from one machine to another. As a result of this decision, they use different versions of Java. (b) D'Agents uses a modified version of an older Sun JVM, whereas NOMADS uses a custom JVM called Aroma (a "clean-room" implementation of the Java VM specification, and mostly JDK 1.2.2 compatible). This decision has a significant impact on performance, because the newer Sun JVM is generally more efficient and supports Just-In-Time (JIT) compilation. The NOMADS JVM is an untuned research prototype with no JIT compiler. Despite its age, the D'Agents JVM has one benefit: optimized string-handling routines that are important for our test application.

Table 1. Relevant features of systems used in our experiments.

Feature	D'Agents	NOMADS	EMAA	KAoS
a) strong/weak	strong	strong	weak	weak (Voyager)
b) JVM version	1.0.2	Aroma	1.2.2	1.3
c) JVMs used	multiple	multiple	one	one
d) what moved	all code, data, stack	data, stack	data	data
e) code caching	no	yes	preinstalled	preinstalled
f) encoding	custom, fat	custom	serialized	serialized
g) communication	sockets	sockets	sockets	sockets (Voyager)
h) socket reuse	no	no	yes	yes
i) security	off	off	off	off

(**c**) For several reasons, D'Agents and NOMADS also create a new JVM process for each incoming agent, while the others simply add a new thread to the existing JVM. This choice raises the cost of jumps in D'Agents and NOMADS.

(**d**) Only D'Agents moved every bit of agent state (all necessary classes, the stack of the jumping thread, and the reachable parts of the heap) on every jump. NOMADS cached the code on the server during the first jump, so subsequent jumps did not need to move code. EMAA and KAoS do not transmit code with an agent; the recipient fetches the code from a class server and then caches it for future use. As a result, in our experiments they essentially never moved code. EMAA and KAoS never move thread-stack state. As a result, EMAA and KAoS agents are relatively small.

(**f**) EMAA and KAoS used Java serialization to encode the agent object, but the other two had their own encoding for agent state. The D'Agents encoding is particularly verbose, increasing the size of its agents. Section 3.1 presents the actual agent sizes from our experiments.

(**g**) Interestingly, none used RMI to move a jumping agent, choosing the more efficient socket mechanism (using TCP/IP). (**h**) D'Agents and NOMADS created a new socket connection for every jump, whereas EMAA and KAoS (really Voyager) "cached" the open socket and re-used it for subsequent jumps, reducing the overhead of jumps.

(**i**) Finally, where security mechanisms like encryption or authentication were available, they were disabled for these experiments. Such features have significant performance impact, but varied so much across systems that we chose to eliminate them from this initial study of the impact of other features.

3   Experiments

Our goal was to compare the scalability of the mobile-agent approach versus the client/server approach in an information-retrieval (IR) task as the number of clients increased. In our experiments, we explore the effect of increasing the number of clients and agents on a single mobile-agent server and its network connection. While our experiments do not always identify the boundaries of the performance space (not all experiments reach the limit of CPU or network

capacity, for all agent systems), the results invite comparison between mobile-agent system designs, and bring some understanding to the structure of the performance space.

We implemented a simple IR task using both an agent and a client/server architecture. Our task filters the results of a simple keyword query on a collection of documents stored at the server. The client/server application downloads all documents resulting from every query, and does its filtering on the client machine. The agent-based application sends an agent to do the filtering on the server and returns with only the matching documents. The client/server application is written in C++ (for speed), while the agent-based application is written in Java (for mobility). The tradeoff is thus between network bandwidth consumption and processing speed, between a fast language on distributed clients and a slower language on a shared server. In our experience, mobile-agent applications often offer this tradeoff, and are particularly interesting in situations where the server does not support the application's needs directly in its RPC interface.

We recognize that this experiment does not explore the full range of mobile-agent capability, in particular, the ability to jump to more than one server, but the scalability of mobile-agent systems even for single-hop applications is not yet well understood. The results of the single-hop experiments presented here are a critical foundation for future research, since even a multi-hop agent must decide whether to jump at all.

3.1   The Experiments

Our IR task involved two steps: a keyword query selected a set of documents from the collection, then a filter procedure scanned the selected documents to return those that contain a given string. Our document server implemented only the keyword-query operation. In our client/server implementation, the selected documents were returned to the client, which ran the filter. In our agent implementation, the agent filtered the documents at the server, then carried the resulting subset back to the client host. This application is representative of the type of computational task that might be used in an agent-based information-retrieval application.

Because the keyword query is common to both implementations, we removed that step, and used a fixed list of sixty 4096-byte documents. Although in both implementations we scan all sixty documents, we chose to declare a certain fraction of the documents to "pass", ignoring the actual results of the filter, to increase our control over the size of the task output. In our experiments the "pass ratio" was either 5% or 20%.

We wrote the client/server applications in C++ using TCP/IP connections with a simple protocol for handshaking between client and server. The total query time is the time recorded at the client host to send the keywords to the server, receive the sixty documents, and filter the sixty documents on the client. We average these times to give average total query time.

Table 2. Comparison of average IR task times.

Pass ratio	C++	D'Agents	EMAA	KAoS	NOMADS
5% ratio	2.92ms	55.9ms	88.9ms	63.5ms	14497ms
20% ratio	3.02ms	61.6ms	96.1ms	73.6ms	14516ms

We wrote the agent application in Java. The speed of any application written in the Java language, even with a JIT compiler, is slower than that of an equivalent implementation in C++. This difference works only in the favor of the client/server approach; any performance benefits seen with the agent approach are not due to language differences. We ported the agent application to each of our four agent platforms, and reviewed the ported code to ensure that they were functionally identical.

There are four different virtual machines used by the four different mobile-agent platforms. D'Agents "AgentJava" uses a modified JDK 1.0.2, EMAA uses the Linux Blackdown JDK 1.2.2 port with JIT compiler (although the agents were compiled with a Java 1.3 compiler), KAoS uses the Sun JDK 1.3.0-02, and NOMADS uses its own JVM that has not yet been optimized for speed (Aroma release 20010327). To understand the speed differences, we ran the IR task alone in each platform.

C++ was markably faster due to inefficient Java file I/O routines. All of the times reflect little actual disk activity because the underlying Linux file cache held all of the documents used. All of the Java tests used the same code, so any difference in performance was due to differences in the JVM or JIT compiler. Due to an optimized string-handling library, D'Agents was significantly faster than EMAA or KAoS, even though it did not use a JIT compiler. These differences accounts for some of the performance differences seen in our scalability tests below.

In our scalability experiments, each client agent looped over many queries. For each query, the agent jumped to the server, obtained the list of sixty documents, ran the filter over those documents, obtained the subset that "pass" the filter, and jumped back. The elapsed time, measured on the client, was the total query time. The agent also measured its time on the server, the "task time." The "jump time" was the difference between the total time and task time.

In our experiments we varied the number of clients (1 to 20, each on a separate machine), the network bandwidth to the server (1, 10, 100 mbps),[1] shared by all clients, and the pass ratio (5% or 20%).

Other parameters, fixed for these experiments, were the number of documents in the collection (60), the document size (4096 bytes), and the number of queries (10-1000 queries, depending on the agent system, using whatever number of queries was required to get repeatable results). The query rate was set to average one query per two seconds, but uniformly distributed over the range 0.25–0.75 queries per second. This randomness prevents agents from exhibiting

---

[1] In this paper we use the prefixes m and k to refer to powers of 10, and the prefixes M and K to refer to powers of 2. Thus 10 mbps refers to 10,000,000 bits per second.

Table 3. Agent sizes in bytes. All were measured "on the wire," including all protocol overhead.

Agent	D'Agents	EMAA	KAoS	NOMADS
5% client to server	16,317	21,641	4,560	8,403
5% server to client	29,311	21,706	17,104	58,959
20% client to server	17,668	59,551	5,380	8,403
20% server to client	68,186	59,817	56,183	210,256

synchronous behavior. This query rate is a maximum: if a query takes longer than two seconds to complete its task, the next query will not be started until the agent returns to its client machine.

The agent size depended on the agent system (Table 3). D'Agents includes all the classes with every agent, so its base agent size is the largest. NOMADS can optionally compress the agent state in transit, but that option was not used in our experiments.

The size of agents going from client to server was incorrectly high in some cases, because our implementation had the same agent jump back and forth to obtain an average performance. After the first trip to the server, EMAA carried the documents *to* the server on every trip, and KAoS and D'Agents carried a small amount of extraneous state information. We expect that the effect on D'Agents and KAoS performance was small, but that EMAA's jump times may have increased significantly. NOMADS encoded the documents with several bytes per character, while other systems used one byte per character. Although this makes the NOMADS agents much larger, the computational overhead of NOMADS dominated its results so the agent size was not much of a factor.

We ran the experiments on a set of twenty-one identical Linux workstations.[2] Twenty of the machines act as clients and one acts as the document server. We interconnected the computers with a 100 mbps Ethernet,[3] but could reduce the bandwidth available by inserting a software bandwidth manager set to 10 mbps or 1 mbps.[4] The network was full duplex at all bandwidths.

4   Results and Discussion

We plot several aspects of the results in a series of figures. We first consider the total query time, and then its components "task time" and "jump time." Then we make a direct comparison between the client/server times and the agent times, by presenting the ratio between client/server and agent times.

The plots are missing some NOMADS points. Also, several of the EMAA points are slightly too low because of early termination of some agents. The general EMAA trends are correct, although little should be interpreted into the

---

[2] VA Linux VarStation 28, Model 2871E, 450 mHz Pentium II, 256 MB RAM, 5400 rpm EIDE disk, running the Linux 2.4.2-2 (RedHat 7.1) operating system.

[3] With a one-way measured throughput of 65 mbps.

[4] DummyNet; see <http://info.iet.unipi.it/˜luigi/ip_dummynet/>.

details of any non-linear wiggles. (Most of the agent systems had trouble in the 10 mbps tests, due we believe to some bugs in RedHat Linux 7.1 or its interaction with dummynet.)

4.1   Total Query Time

Each plot in Figure 1 shows the averaged per-query time, in milliseconds, for all systems (note there is a separate scale for the NOMADS data).

The figure shows six plots, for three bandwidths (1, 10, and 100 mbps) and two pass ratios (5% or 20%). Generally speaking, any implementation will slow down linearly with the number of clients, due to increasing contention for the network and the server's CPU. A query time exceeding 2000 milliseconds indicates that the clients have failed to sustain the desired query rate and have slowed to match the system's capacity.

The slope of the line depends on the overhead of that implementation, the parameters of the query, and the speed of the network and CPU. An inflection point, where the slope suddenly increases, indicates that the load exceeded the limitations of the network or CPU. That effect can be seen most readily in our 10 mbps client/server experiments, where the demands of 12 clients exceed the limits of the network.

In the 1 mbps network, the fact that agents bring back only 5 or 20% of the documents allows them to be less sensitive to the constraints of the slow network, while the client/server approach is bandwidth-limited. Here, as in the 10 mbps network, D'Agents, EMAA, and KAoS clearly perform much better than client/server. NOMADS is much slower, due to its slower JVM (as we discuss in the next section).

In the 100 mbps network, however, client/server is the clear winner. In this environment, the network has more than enough bandwidth to allow the clients to retrieve all of the documents. With the network essentially free, the slower computation of the agents (using Java rather than C++, and sharing the server rather than dispersing among the clients) makes the mobile-agent approach a less attractive option.

The differences between mobile-agent systems are better examined in terms of the task times and jump times.
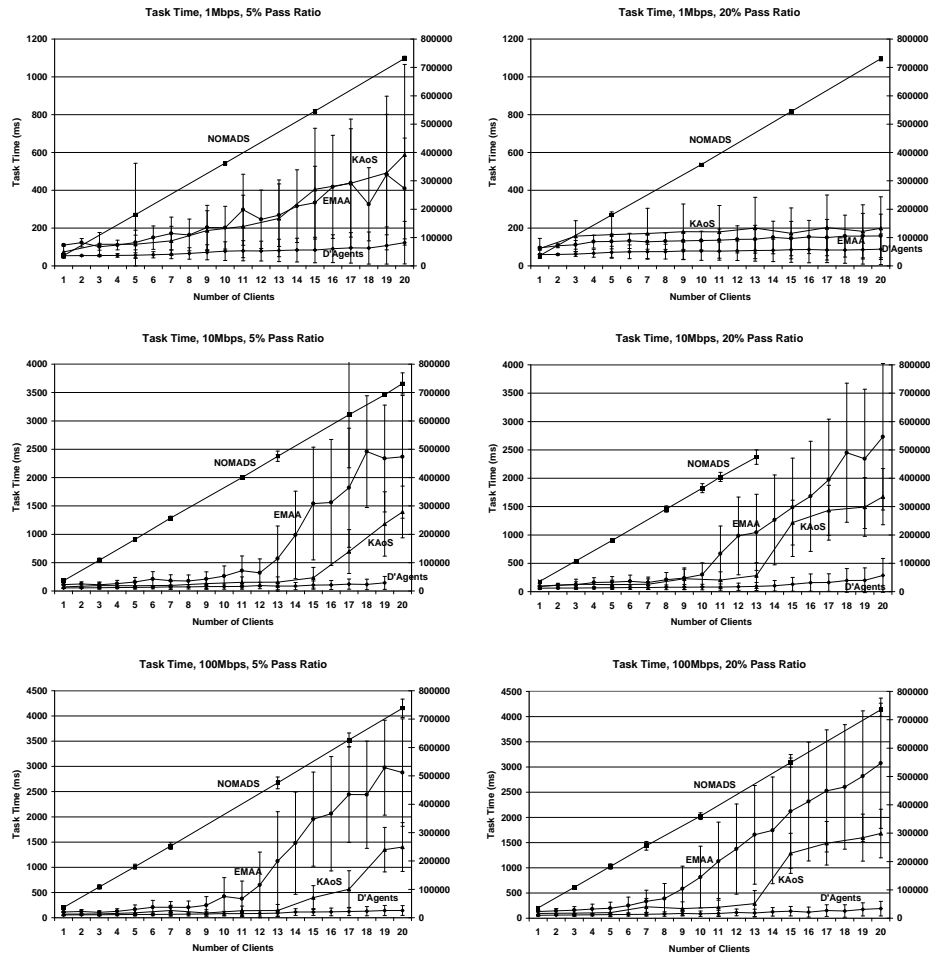
4.2   Task Times

Each plot in Figure 2 shows the task time for all agent systems. The task time is the time for computation of the filtering task only. Recall, however, that a client will not generate a query until its previous query has finished. In a network-limited configuration the query rate is reduced, reducing load on the server. Thus, the task times do depend on the bandwidth of the network.

The most notable feature in these graphs is the dramatic difference between the NOMADS times (which have a separate y-axis scale) and the other agent systems. This difference is due to the home-grown Aroma JVM used in NO-MADS, which has not been tuned. The NOMADS data grows linearly with the
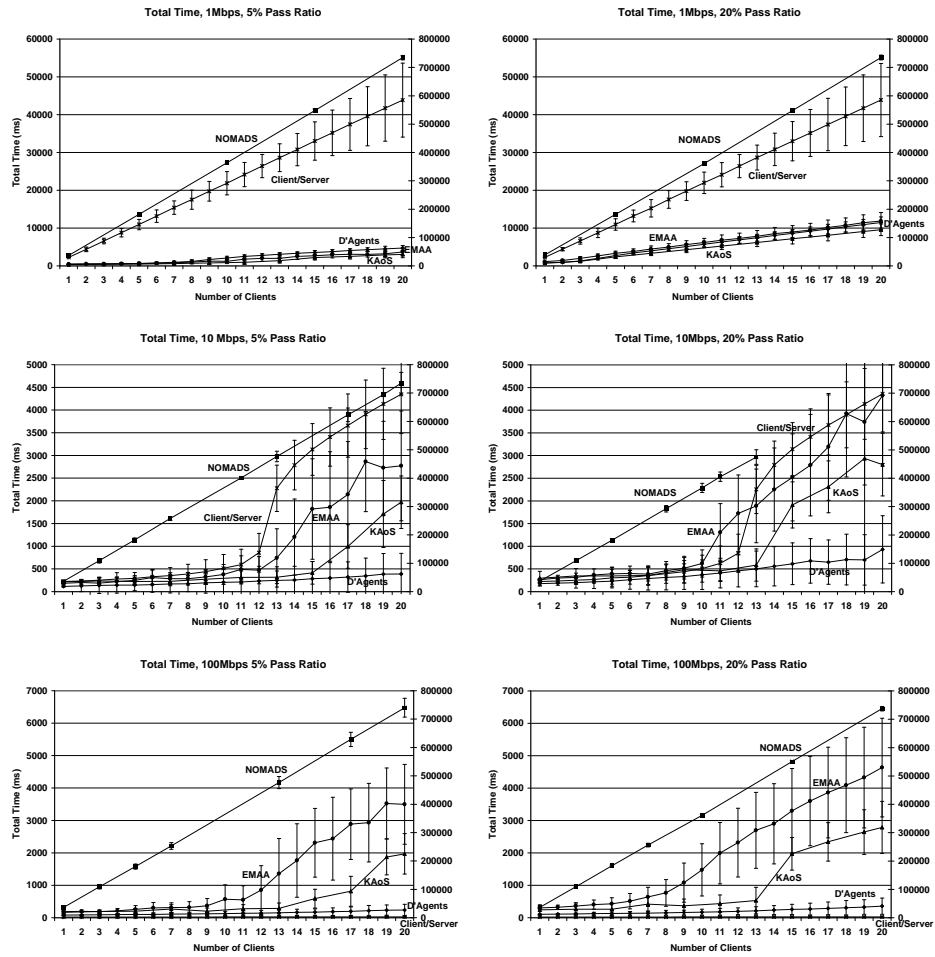
Fig. 1. Total query times, for all systems, all three bandwidths, and both pass ratios. We show error bars indicating one standard deviation from the mean. Note that the vertical scale varies. The NOMADS data should all be read using the scale on the right-hand side of the graph.

Fig. 2. Task times, for all systems, all three bandwidths, and both pass ratios. We show error bars indicating one standard deviation from the mean. Note that the vertical scale varies. The NOMADS data should all be read using the scale on the right-hand side of the graph.

number of clients, indicating that the server's CPU is always the limiting factor for NOMADS.

The D'Agents task time is the fastest, in large part because it uses an *older* version of the JVM, with native (rather than Java) implementations of the critical string-manipulation routines. Our document-scanning application stresses those routines, leading to better performance for D'Agents.

The D'Agents time is largely constant, because the query rate is low enough to not stress the server CPU. In the 1 mbps network, the EMAA 5% tests begin to overload the server at about 8 clients, whereas the EMAA 20% tests overload the network and thus never overload the server. In the 10 mbps network, EMAA overloads the server CPU after about 10 (20%) or 12 (5%) clients. EMAA is slower than KAoS, largely due to a different version of the JVM (recall Table 2).
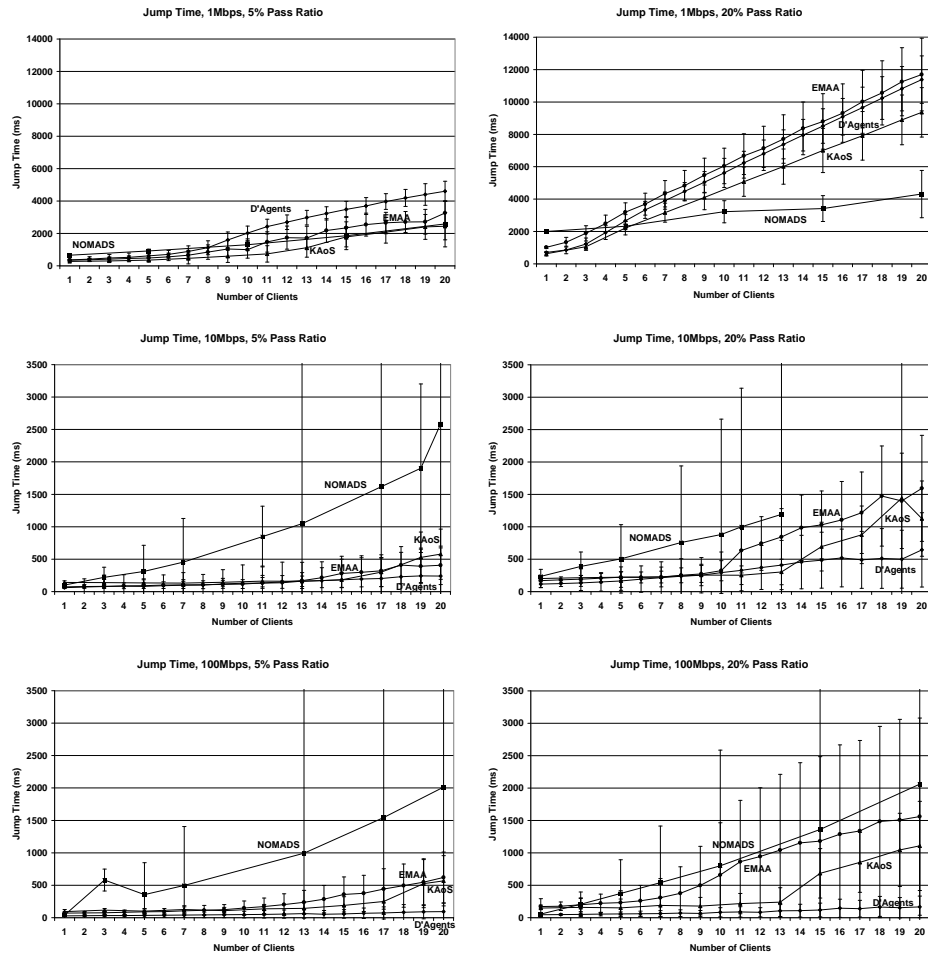
4.3   Jump Times

Each plot in Fig. 3 shows the average per-query jump time for each system. Recall that the jump time is the total query time minus the task time, so it includes all of the computational overhead of a jump (serialization, network protocol, deserialization, and reinstating an agent) as well as the network time.

The jump times are most difficult to interpret, because they depend on the load of both network and server. The higher NOMADS times in fast networks, for example, are likely due to the heavy load on the CPU impacting the time needed for serialization, transmission, and deserialization of jumping agents. Note that NOMADS had the fastest jump times in the most congested network (1mbps at 20% pass ratio and 20 clients), despite having the largest agents.

In slow 1 mbps networks, we expect that systems with smaller agents (like EMAA and KAoS) jump faster than systems with bigger agents (like NOMADS and D'Agents). The results in the top row of Figure 3 are therefore surprising. NOMADS was fast, indeed sometimes fastest by far; the reason is that NOMADS task times were so large that agents only occasionally cross the network, and the network never experiences congestion or heavy load. EMAA was slower than KAoS because the net effect of carrying its payload in both directions was that EMAA's agents were considerably larger.

In the 1 mbps case, the network was the bottleneck; in the 5% graph we can see D'Agents, EMAA, and KAoS change slope when they first encounter that bottleneck. In faster networks, the server's load was the bottleneck. Again, we can see inflection points where D'Agents, EMAA, and KAoS first encounter that bottleneck. NOMADS was computation-bound in all cases.

It is difficult to attribute specific design decisions to the jump times measured in our experiments. Clearly it was helpful to have smaller agents, but even in the slowest network we found that the computational overhead was often a determining factor in the time required for a jump.

Fig. 3. Jump times, for all systems, all three bandwidths, and both pass ratios. We show error bars indicating one standard deviation from the mean. Note that the vertical scale varies.

4.4   Ratio of Client/Server Time to Agent Time

Each plot in Figure 4 shows the "performance ratio," which is the client/server query time divided by the mobile-agent query time. A ratio of 1 indicates that the agent approach and the client/server approach are equivalent in performance; higher than 1 indicates that agents were faster. The NOMADS ratios are indistinguishable from zero because their times were so large. For the other three systems, there are three different effects, dependent on bandwidth.

In the 1 mbps curves, we see that the performance ratios climbed, and then fell or level off. For small numbers of agents, the performance ratio improved quickly because the client/server approach was bandwidth limited, while the agent approach was not. With a few more agents, it reached the network bandwidth limit and became slower, reducing the performance ratio. Once both client/server and agent performance reached the same slope, the performance ratio leveled off. In the 20% case, the ratio was about 4–5, which is reasonable considering that the agents moved 1/5th of the data, but with some overhead. In the 5% case, where the agents moved 1/20th of the data across the network, the ratio was 8–15.

In the 10 mbps curves, we see a different effect. Here, the agents never hit the network limit, but the client/server implementation hit the limit at 12–13 clients. The performance ratio suddenly improved. The performance ratio for KAoS and EMAA then dropped, due to increasing server load.

In the 100 mbps curves, all performance ratios were low and declined steadily as more clients were added, due to increasing contention for the server's CPU.

5   Related work

Researchers have developed numerous mobile-agent systems over the past decade. Few papers, however, present any substantial study of system performance. Fewer still examine scalability. We discuss the most relevant papers here.

Ismail and Tichy [9] compare the performance of client/server (RMI) with mobile agents. The client contacts one server to obtain a list of hotels, and another server to obtain a phone number for each hotel (one a time). In the alternative implementation a mobile agent visits the first server and then the second server, returning with all phone numbers for all candidate hotels. Although the agent is a multi-hop agent, unlike those in our study, the application is analogous. Mobile agents provide a performance advantage when the agent retrieves a sufficient number of candidate hotels. In their study, however, there was only a single client and a single agent, and no other load on the servers.

Johansen [10] used an application like ours, though using images or video files rather than text documents. The results are directly analogous to our own results, with similar crossing points in the mobile-agent vs. client-server performance curves. They do not, however, study the scalability of the server, since there is one client sending one agent to the server.

Straßer and Schwehm consider an abstract application, using an analytic model and parameters derived from the Mole mobile-agent platform [13]. They
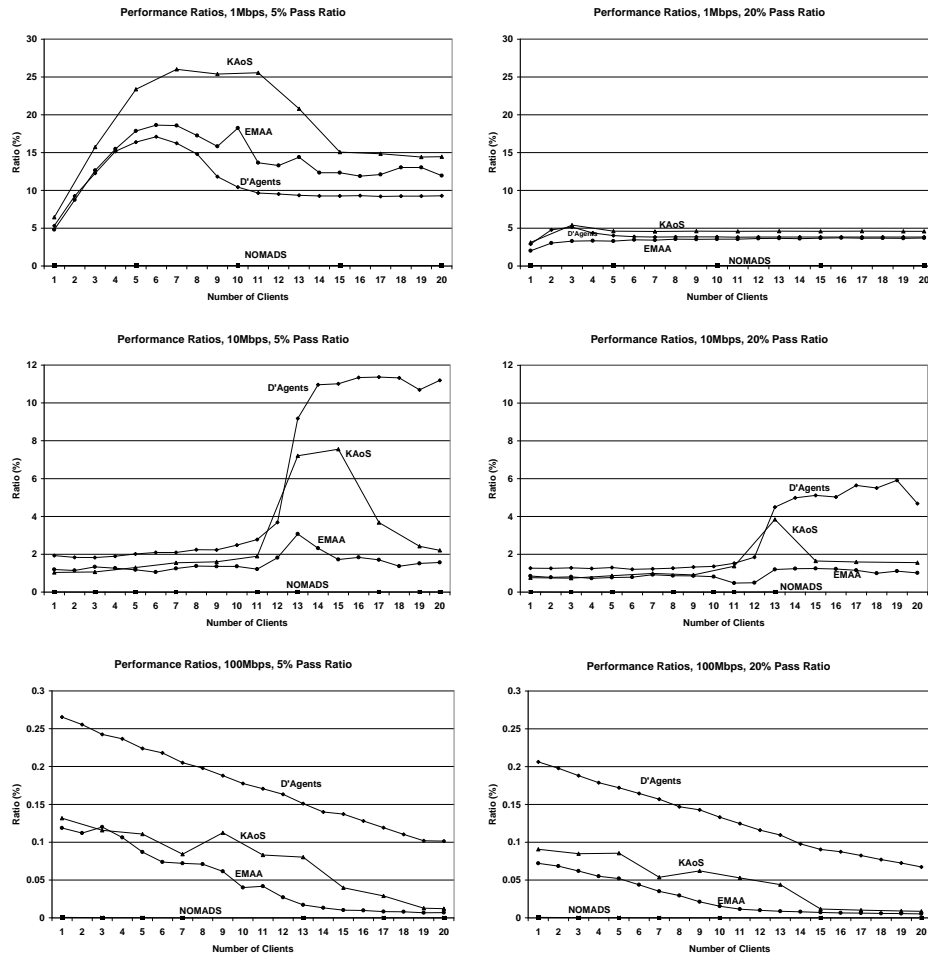
Fig. 4. Performance ratios for all systems, for both pass ratios, combining all systems on one plot. Note that the vertical scale varies.

consider only a single mobile agent, although it may visit multiple servers. They have limited experiments on only one mobile-agent system, and they do not evaluate the scalability of an agent server.

Küpper and Park use an analytic model, parameterized by a small experiment, to predict the scalability of a telecommunications application [11]. They compare two approaches: stationary agents, in which the call-setup code for a user is always resident in the user's *home* network, and mobile agents, in which the call-setup code moves to the user's current network. Mobile agents lead to improved call-setup times as long as the user makes enough calls before moving to a new network. Their paper does not measure real mobile-agent systems, nor study the scalability of a mobile-agent server.

Baldi and Picco also use an analytic model, parameterized by experiments, to examine a different aspect of scalability [1]. They compare the network traffic generated by a variety of approaches (including client/server and mobile agents) for collecting statistics from a distributed set of network devices. They conclude that mobile agents (in general, mobile code) can reduce network traffic, relative to a client-server approach, and thus allow their application to support a larger number of devices. They consider only a single mobile agent. They compare only network traffic in bytes and no measures of time. They do not consider the scalability of the mobile-agent platform itself.

Theilmann and Rothermel also examine the performance of a mobile-agent application that visits many data servers to filter data [15]. The client/server approach downloads all data for filtering at the client. In the mobile-agent approach, one mobile agent is dispatched as *close* to each data server as possible. They achieve significant cost savings whether they used hop counts or round-trip time as basis for measuring distance. "Cost" seems to relate to total number of bytes transferred across the network. As in our study, the cost savings depend entirely on the amount of data examined (and filtered out) on each remote host. They do not study scalability of the agent servers, however, since there is only one agent sent to each data server.

Woodside [16] proposes a model for scalability and analysis of mobile-agent systems. The paper examines scalability in terms of the time for a mobile agent to complete a "tour" (an execution that involves visiting several hosts) as the number of hosts (agent servers) increases with a corresponding increase in the number of agents. The model presented does not account for communication costs, one of the central factors in our study. Also, the paper does not provide any experimental results.

Dikaiakos and Samaras [6, 5] develop a framework for performance analysis of mobile-agent systems. They propose three layers of benchmarks: micro-benchmarks that test individual operations such as messaging and migration, micro-kernels that are small, synthetic tasks that would be part of typical applications, and application-kernels that use actual application-level functionality and workloads. They present experimental results for three real mobile-agent platforms using micro-benchmarks and micro-kernels that describe performance in terms of throughput for a single agent, but do not address scalability.

The consistent theme of previous work, confirmed by our own work, is that a mobile-agent approach outperforms a client-server approach as long as the application involves analysis of enough information, and enough reduction of the data returned to the client, to outweigh the overhead of sending the mobile agent in the first place. Our study is unique in its study of a server heavily loaded by mobile agents from multiple clients, and unusual in its cross-comparison of several mobile-agent systems.

6   Conclusion

In our experiments we found that the scalability of mobile-agent systems, in comparison to an alternative client/server implementation of the same application, depends on many different environmental parameters. Overall, the four mobile-agent systems we examined scale reasonably well from 1 to 20 clients, but when we compare them to each other and to a client/server implementation they differ sometimes dramatically. The client/server implementation was highly dependent on sufficient network bandwidth. The agent implementations saved network bandwidth at the expense of increased server computation.

The performance of NOMADS clearly suffered from the untuned virtual machine. The relative performance of the other three mobile-agent systems varied, depending on the mix of computation and network in the application, reflecting their different mix of overheads. The optimized string functions in the D'Agents JVM helped prevent server overload when the network was fast. The smaller agents of KAoS and EMAA were an advantage in slower networks, although the cost of serialization hurt EMAA.

Our experiments are admittedly only a first step toward understanding the performance of, and scalability of, mobile-agent systems. These results are for a single application, in which mobile agents hop once to the server and once back to the client. The application exercises string processing on the server, and the transportation of documents in a jumping agent, but does not exercise agent-agent communication, security mechanisms, multi-hop mobile agents, or complex network topologies. The relative performance of our four mobile-agent systems depends in part on the current state of their implementations. Indeed, it is difficult to tease out the performance effects of the design differences outlined in Table 1, because their effects were confounded.

It is clear from our results that mobile agents can be beneficial in situations with low network bandwidth and plentiful server capacity. Indeed, in many environments it is easier to add more server capacity than to add network capacity, particularly those with wireless networks. For applications demanding high performance and scalability to hundreds or thousands of active agents, further research is necessary to develop light-weight agent systems and scalable agent platforms. We are investigating automated ways to build parallel or distributed mobile-agent servers and services.

References

1. M. Baldi and G. P. Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In *Proc. of the Twentieth International Conference on Software Engineering*, pages 146–155, Kyoto, Japan, April 1998.
2. J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley. KAoS: Toward an industrial-strength open agent architecture. In J. Bradshaw, editor, *Software Agents*, pages 375–418. AAAI/MIT Press, 1997.
3. J. M. Bradshaw, N. Suri, A. K. Cañas, R. Davis, K. Ford, R. Huffman, R. Jeffers, and T. Reichherzer. Terraforming Cyberspace. *IEEE Computer*, 34(7), July 2001.
4. D. Chacón, J. McCormick, S. McGrath, and C. Stoneking. Rapid application development using agent itinerary patterns. Technical Report #01-01, Lockheed Martin Advanced Technology Laboratories, March 2000.
5. M. Dikaiakos, M. Kyriakou, and G. Samaras. Performance evaluation of mobile-agent middleware: A hierarchical approach. In *Proc. of the Fifth IEEE International Conference on Mobile Agents*, LNCS, Atlanta, GA, December 2001. Springer-Verlag.
6. M. D. Dikaiakos and G. Samaras. A performance analysis framework for mobile-agent systems. In *Infrastructure for Agents, Multi-Agents, and Scaleable Multi-Agent Systems*, volume 1887 of *LNCS*, pages 180–187. Springer-Verlag, 2001.
7. R. Gray. *Agent Tcl: A flexible and secure mobile-agent system*. PhD thesis, Dept. of Computer Science, Dartmouth College, June 1997. Available as Dartmouth Computer Science Technical Report TR98-327.
8. R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. D'Agents: Security in a multiple-language, mobile-agent system. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *LNCS*, pages 154–187. Springer-Verlag, 1998.
9. L. Ismail and D. Hagimont. A performance evaluation of the mobile agent paradigm. *ACM SIGPLAN Notices*, 34(10):306–313, October 1999.
10. D. Johansen. Mobile agent applicability. In *Proc. of the 2nd Int'l Workshop on Mobile Agents*, volume 1477 of *LNCS*, pages 80–98, Stuttgart, Germany, September 1998. Springer-Verlag.
11. A. Küpper and A. S. Park. Stationary vs. mobile user agents in future mobile telecommunication networks. In *Proc. of the 2nd Int'l Workshop on Mobile Agents*, volume 1477 of *LNCS*, pages 112–123, Stuttgart, Germany, September 1998. Springer-Verlag.
12. S. McGrath, D. Chacón, and K. Whitebread. Intelligent mobile agents in the military domain. In *Proc. of the Autonomous Agents 2000 Workshop on Agents in Industry*, Barcelona, Spain, 2000.
13. M. Straßer and M. Schwehm. A performance model for mobile agent systems. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume II, pages 1132–1140, Las Vegas, July 1997.
14. N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, and R. Jeffers. Strong mobility and fine-grained resource control in NOMADS. In *Proc. of the Second Int'l Symp. on Agent Systems and Applications and Fourth Int'l Symp. on Mobile Agents (ASA/MA2000)*, volume 1882 of *LNCS*, pages 2–15, Zurich, Switzerland, September 2000. Springer-Verlag.
15. W. Theilmann and K. Rothermel. Optimizing the dissemination of mobile agents for distributed information filtering. *IEEE Concurrency*, 8(2), April–June 2000.
16. M. Woodside. Scalability metrics and analysis of mobile agent systems. In *Infrastructure for Agents, Multi-Agents, and Scaleable Multi-Agent Systems*, volume 1887 of *LNCS*, pages 234–245. Springer-Verlag, 2001.