Performance Evaluation of Mobile Agents for E-Commerce Applications

Rahul Jha and Sridhar Iyer

Kanwal Rekhi School of Information Technology, Indian Institute of Technology Bombay, Powai, Mumbai - 400 076.

{rahul, sri}@it.iitb.ac.in
http://www.it.iitb.ac.in

Abstract. Mobile agents are emerging as a promising paradigm for the design and implementation of distributed applications. While mobile agents have generated considerable excitement in the research community, they have not translated into a significant number of real-world applications. One of the main reasons for this is the lack of work that quantitatively evaluates (i) the effectiveness of one mobile agent framework over another, and (ii) the effectiveness of mobile agents versus traditional approaches. This paper contributes towards such an evaluation. We identify the underlying mobility patterns of e-commerce applications and discuss possible client-server (CS) and mobile agent (MA) based implementation strategies for each of these patterns. We have compared the performance of three mobile agent frameworks, viz., Aglets, Concordia and Voyager, for their suitability for an e-commerce application development. We have performed experiments to quantitatively evaluate the effectiveness of CS versus MA strategies, for the identified mobility patterns. We used Java sockets for the CS implementation and the ObjectSpace VoyagerTM (Voyager) framework for the MA implementation. In this paper, we present our observations and discuss their implications.

1 Introduction

The emergence of e-commerce applications has resulted in new net-centric business models. This has created a need for new ways of structuring applications to provide cost-effective and scalable models.

Mobile Agent (MA) systems have for some time been seen as a promising paradigm for the design and implementation of distributed applications. A mobile agent is a program that can *autonomously migrate* between various nodes of a network and perform computations on behalf of a user. Some of the benefits provided by MAs for structuring distributed applications include reduction in network load, overcoming network latencies, and support for disconnected operations [10]. The use of MAs has been explored for a variety of applications such as information retrieval, workflow management systems and e-commerce applications [7, 11, 12, 13].

While there are some experimental as well as commercial deployments of MAs for e-commerce applications (e.g. MAgNET [3], Gossip [www.tryllian.com]), they have still not translated into a significant number of real-world applications. We believe that one of the main reasons for this is the lack of work that quantitatively evaluates (i) the effectiveness of one mobile agent framework over another, and (ii) the effectiveness of mobile agents versus traditional approaches. This paper contributes towards such an evaluation.

We identify the underlying mobility patterns of e-commerce applications and discuss possible client-server (CS) and MA based implementation strategies for each of these patterns. We identify various application parameters that influence performance, such as, size of CS messages, size of the MA, number of remote information sources, etc., and have performed experiments to study their effect on application performance. We have used the *user turnaround time* (time elapsed between a user initiating a request and receiving the results), as the metric for performance comparison. We have performed experiments for MA using Aglets, Concordia and Voyager frameworks and Java socket for CS implementation. In this paper, we present our observations and conclusions regarding the choice of an agent framework and implementation strategy, for an e-commerce application.

While there exist some qualitative and quantitative studies of MAs in other application domains [2, 5, 7, 12, 13], to the best of our knowledge, there is no literature on the performance evaluation of MAs in the domain of e-commerce. Also, there is no prior work on identifying mobility patterns for e-commerce applications, or on evaluating mobile agent frameworks for these mobility patterns.

Section 2 provides a classification of identified mobility patterns, and section 3 discusses various implementation strategies for these mobility patterns. Section 4 describes the experimental setup. Section 5 presents a qualitative and quantitative evaluation of the three MA frameworks chosen for our study. Section 6 presents quantitative evaluation experiments of mobile agent and client-server implementation strategies. Section 7 concludes with a discussion on the implications of using mobile agents for e-commerce applications.

2 Mobility Patterns for Mobile Agents

Mobility in MAs can be characterized by the set of destinations that an MA visits, and the order in which it visits them. Hence we identify the following parameters to characterize the mobility of an MA:

1. **Itinerary :** The set of sites that an MA needs to visit. This could either be *static* (fixed at the time of MA initialization), or *dynamic* (determined by the MA logic).
2. **Order :** The order in which an MA visits the sites in its itinerary. This may also be either *static* or *dynamic*.

Based on these parameters, we distinguish MA applications in e-commerce as possessing one of the following mobility patterns:

Static Itinerary (SI)

The itinerary of the MA is fixed at the time of initialization and does not change during execution of the application. We further distinguish such applications into:

- **Static Itinerary Static Order (SISO)**
  The order in which an MA visits the sites in its itinerary is static and fixed at the time of initialization. An example application is that of an auction MA, which may be required to visit a set of auction sites in a pre-specified order.
- **Static Itinerary Dynamic Order (SIDO)**
  The order in which an MA visits the sites in its itinerary is decided dynamically by the MA. An example application is that of a shopping MA which may visit a set of online shops, in arbitrary order, to find the minimum price for a product.

Dynamic Itinerary (DI)

The itinerary of the MA is determined dynamically by the MA itself. Obviously, at least the first site in the itinerary needs to be fixed at the time of initialization. An example application is that of a shopping MA that is required to find a particular product. A shop that does not have the product may recommend an alternative shop, which is included in the MA's itinerary dynamically. It may be noted that dynamic itinerary always implies dynamic order.

3 Implementation Strategies for Applications

We identify four implementation strategies that may be adopted by a typical e-commerce application:

1. **Sequential CS :** This is based on the traditional client-server paradigm. The client makes a request to the first server and after processing the reply, makes a request to the second server and so on, till the list of servers to be visited is exhausted. This strategy is illustrated in figure 1(a).
2. **Sequential MA :** In this case, a single MA moves from its source of origin (client) to the first site (server) in its itinerary. It performs the processing at the server and then moves to the next site and so on, till it has visited all the sites in its itinerary. This strategy is illustrated in figure 1(b).
3. **Parallel CS :** This is also based on the client-server paradigm. However, instead of sequential requests, the client initiates several *parallel* threads of execution, each of which concurrently makes a request to one of the servers and processes the reply. This strategy is illustrated in figure 1(c).
4. **Parallel MA :** In this case, the client initiates multiple MAs, each of which visits a subset of the servers in the itinerary. The MAs then return to the client and collate their results to complete the task. This strategy is illustrated in figure 1(d).

It is also possible to use combinations of the above strategies. In our experiments, we restrict ourselves to these four strategies only.

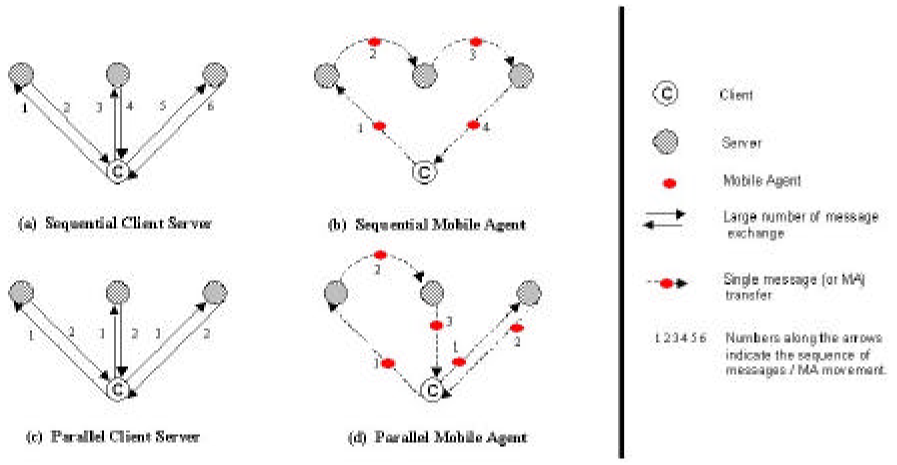
(a) Sequential Client Server
(b) Sequential Mobile Agent
(c) Parallel Client Server
(d) Parallel Mobile Agent
C
Client
Server
Mobile Agent
Large number of message exchange
Single message (or MA) transfer
1 2 3 4 5 6
Numbers along the arrows indicate the sequence of messages / MA movement.

Fig. 1. Implementation Strategies

The feasibility of these implementation strategies is dependent on the mobility patterns of an e-commerce application identified in section 2.We have quantitatively evaluated all the strategies identified in this section for an e-commerce application scenario. The next section describes the chosen e-commerce application, the experimental setup and the performance metric for our performance evaluation experiments.

4 Experimentations

For our performance evaluation experiments, we have chosen a typical e-commerce application, viz., that of *product discovery*. In this application, a single client searches for information about a particular product from the catalogs of several on-line stores.

Our experiment focuses on scenarios where the client requires a highly customized, non-standard search, which the on-line store does not support. This would require the client to fetch a relevant subset of the catalog and implement a search at its end. A product search of this kind could be a "Java powered" wristwatch with "ETA quartz movement". "Java powered" being a non-standard feature, would result in all watches, which match rather standard "ETA quartz movement" featured being fetched to the client's end. The amount of data fetched by the client further increases when multiple shops are searched.

We have implemented such an application using all four strategies mentioned in Section 3 and have evaluated the performance of these implementation strategies.

4.1 Experimental Setup

The experiments were carried out on P-III, 450 MHz workstations connected through a 10 Mbps LAN with typical academic environment loads. We have implemented the MAs using three MA frameworks: Voyager, Aglets and Concordia. The CS implementation consisted of Java client and server applications, using sockets for communication. We have compared the performance of these implementations on the basis of the following parameters:

– number of stores (varies from 1 to 26);
– size of catalog (varies from 100 KB to 1 MB);
– size of client-server messages (varies proportionately with catalog size);
– processing time for servicing each request (varies from 20 ms to 1000 ms);
– network latency (typical academic load on 10Mbps LAN);

Our performance metric is the *user turnaround time*, which we define as the time elapsed between a user initiating a request and receiving the results. This includes the time taken for agent creation, time taken to visit/collect catalogs and the processing time to extract the required information.

The following section presents our qualitative and quantitative evaluation results for the three chosen MA framework.

5 Choice of Mobile Agent Framework

We have performed qualitative and quantitative performance evaluation across three Java based mobile agent frameworks viz., IBM's Aglets [1, 9], Mitsubishi's Concordia [14] and Object Space's Voyager [6]. These MA frameworks provide support for basic agent behaviors such as agent creation, agent dispatch, agent disposal, agent arrival, agent cloning and agent communication [5, 9]. All of the three MA frameworks studied implement weak agent mobility and uses object serialization for agent transfer from one node to another.

5.1 Qualitative Evaluation

Table 1 presents a comparison of Aglets, Concordia and Voyager across some of the mobility features of an MA framework, which influence application's performance.

Features	**Aglets**	**Concordia**	**Voyager**
Multicast support	No	No	Yes
Agent persistence support	No	Yes	Yes
Remote agent creation support	No	No	Yes
Proxy update on MA migration	Yes	No	Yes
Messaging modes between MA	Oneway, synchronous, future	Events	Oneway, synchronous, future
Java messages to MA	Transparent	No	No
Garbage collection	No	No	Yes

Table 1. Qualitative comparison of agent frameworks

5.2 Quantitative Evaluation

We have performed performance evaluation across these three mobile agent frameworks to examine agent messaging and agent migration costs for the product discovery application.

- **Message transfer cost:** A message of fixed size was sent across two nodes on the same LAN and the user turnaround time for this was measured. This was done while varying the number of message exchanges, from 10 to 500.

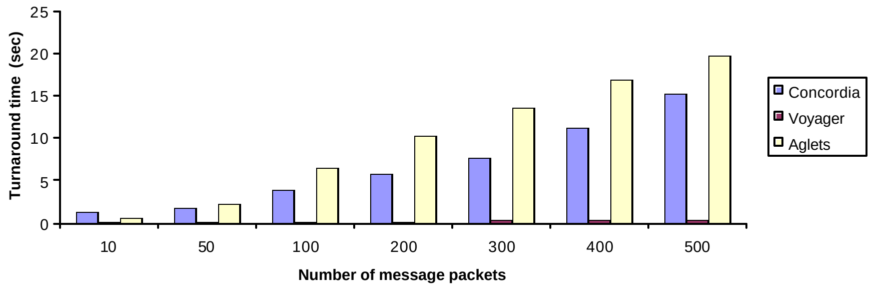

Fig. 2. Message exchange cost for Aglets, Concordia and Voyager

- **Agent shipment cost:** The processing delay at the server was kept constant and the user turnaround time of each of the MA framework was measured for different scenarios of product discovery from 1 to 26 shops.

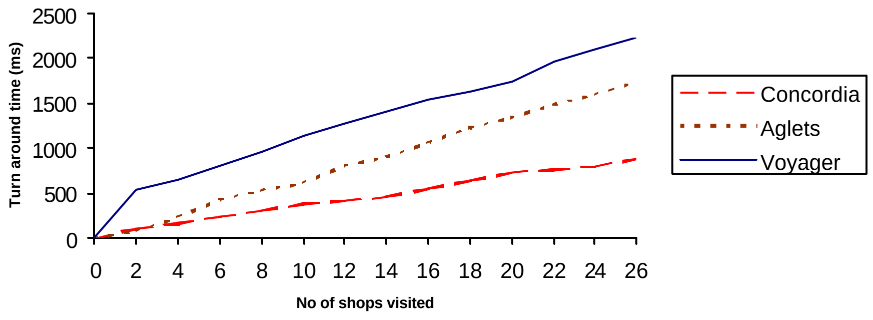

Fig. 3. Agent shipment cost for Aglets, Concordia and Voyager

5.3 Observations

The results of our MA framework evaluation experiments are shown in Fig. 2 and Fig.3. Some key observations are:

– Voyager had the least message transfer cost, followed by Concordia (Fig. 2).

– Concordia had the least agent shipment cost followed by Aglets (Fig. 3).
– Aglets performed better than Concordia when the number the number of messages was less than 50; beyond that Concordia performed better (Fig. 2).
– Aglets performed better when the number of shops to be visited were less than 4; beyond that Concordia showed better performance (Fig. 3).

Our experiments provide directions towards choice of an agent framework. The performance results depend upon the internal implementations of agent frameworks. It would be a useful exercise to study the implementations of these frameworks to identify the performance bottlenecks. However, at present the internals of Voyager and Concordia frameworks are not available for verification and study.

6 Quantitative Evaluation of MA versus CS

We use the same e-commerce application setup (as described in Section 4) for quantitative evaluation of mobile agent and traditional CS approaches. The CS implementation consists of a server that sends a catalog on request and a multi-threaded client that requests a catalog from one or more servers. The client and the server have been implemented using Java sockets. Voyager framework was used for MA implementation of these evaluation experiments. We have performed experiments to determine:

- **Effect of catalog size on turnaround time:** The processing delay at the server was kept constant and catalog sizes of 100KB, 200KB, 500KB and 1MB were used. This was done for different scenarios of product discovery from 1 to 26 shops.

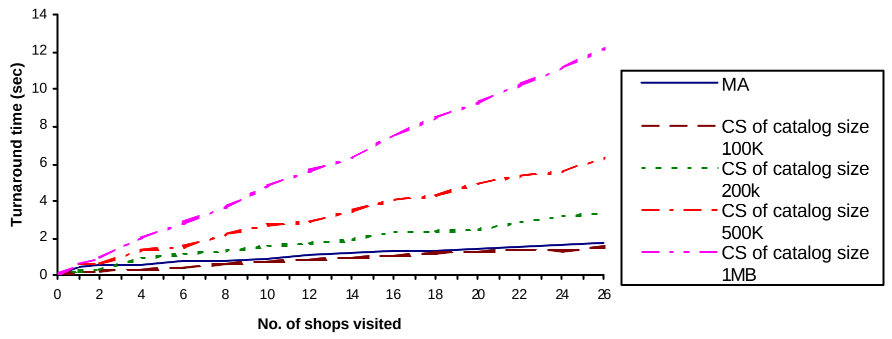

Fig. 4. Effect of catalog size on turnaround time for sequential MA & sequential CS

- **Effect of server processing delay on turnaround time:** The catalog size was kept constant at 1MB and the server delay was varied from 20ms to 500ms to 1000ms. The user turnaround time was measured for different scenarios of product discovery from 1 to 26 shops.

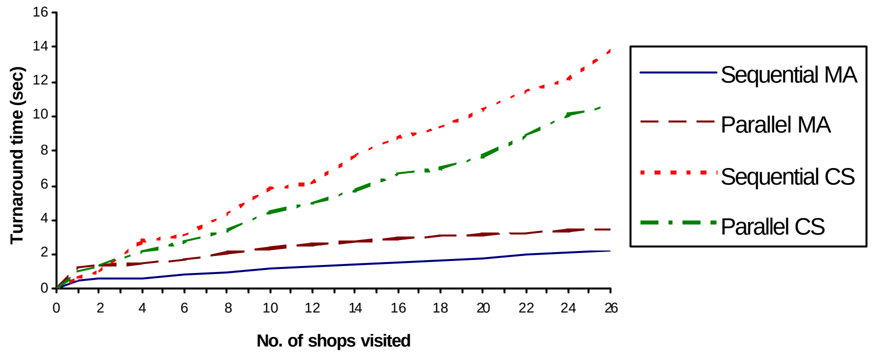

Fig. 5. Turnaround time for a processing delay of 20 ms (catalog size of 1 MB)

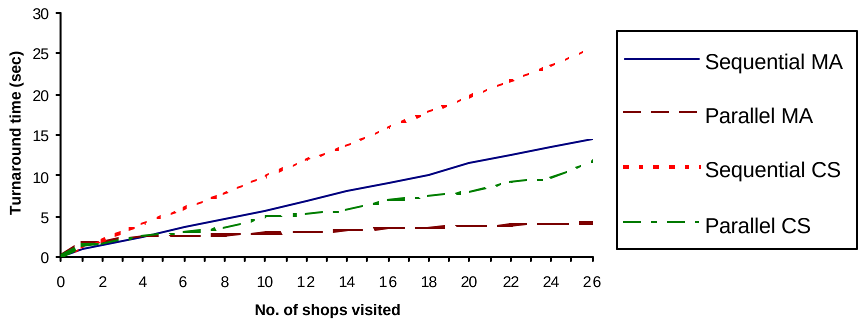

Fig. 6. Turnaround time for a processing delay of 500 ms (catalog size of 1 MB)

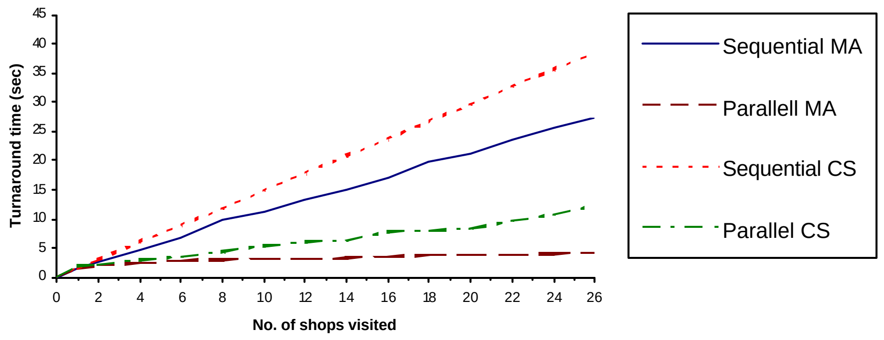

Fig. 7. Turnaround time for a processing delay of 1000 ms (catalog size of 1 MB)

6.1 Observations

The results of our MA versus CS evaluation experiments are shown in the graphs of Fig 4 to 7. Some key observations are:

– The performance of MA remains the same for different catalog sizes while the performance of CS degrades with increase in catalog size (Fig. 4).
– CS implementations perform better than MA implementations for catalog sizes less than 100 KB (Fig. 4).
– MA performs better than CS when the catalog size is greater than 200KB and number of shops to visit is greater than or equal to 3 (Fig. 4).
– MA performs better than all other strategies, for small processing delays (20 ms) and large (1MB) catalog size (Fig. 5).
– Parallel implementations perform better than sequential implementations when the number of shops to visit is greater than or equal to 6 and the processing delay is greater than or equal to 500ms (Fig. 6).
– Parallel MA performs better than parallel CS for higher processing delays (1000ms) and large (1MB) catalog size (Fig. 7).
– *Performance crossover points* i.e. parameter values for which MA starts performing better than CS implementation can be found for a given set of e-commerce application parameters (Fig. 4, 5, 6, 7).

7 Conclusions

We have identified the underlying mobility patterns of e-commerce applications and discussed possible implementation strategies for these patterns using the client-server and mobile agent paradigms. We have evaluated the agent shipment and messaging cost for three chosen MA frameworks and have performed experiments to evaluate the different implementation strategies for MA and CS implementations.

Our experiments suggest that CS implementations are suitable for applications where a "small" amount of information (less than 100 KB) is retrieved from a "few" remote servers (less than 4), having "low" processing delays (less than 20ms). However, most real-world e-commerce applications require large amount of information to be retrieved and significant processing at the server. MA's scale effectively as the size of data to be processed and the number of servers the data is obtained from increase. Scalability being one of the needs of net-centric computing, we find that MAs are an appropriate technology for implementing e-commerce applications.

Parallel implementations are effective when processing delay (greater than 1000ms) contributes significantly to the turnaround time. Our experiments also identify *performance crossover points* for different implementation strategies; this could be used to switch between implementations for performance critical applications. We feel that a *hybrid model* employing all the four identified implementation strategies would result in high performance gain for large-scale distributed applications.

Our experience suggests that mobility patterns play an important role in deciding the implementation strategy to be used for performance critical applications. The selection of an implementation strategy from those feasible for a given application could be based on several criteria such as ease of implementation, performance, availability of technology, etc. We are in the process of carrying out further experiments towards identifying the "ideal" implementation strategy given an application's characteristics.

References

1. Y.Aridov and D.Lang, "Agent design patterns: element of agent application design". In proceedings of Autonomous Agents 1998, pp108-115. ACM, 1998.
2. Antonio Carzaniga, Gian Pietro Picco and Giaovanni Vigna, "Designing Distributed Applications with Mobile Code Paradigms", Proceedings of the 19th International Conference on Software Engineering (ICSE '97) , pp. 22 - 32, ACM Press, 1997.
3. P. Dasgupta, N. Narasimhan, L.E. Moser and P.M. Melliar-Smith, "MAgNET: Mobile Agents for Networked Electronic Trading", IEEE Transactions on Knowledge and Data Engineering, Special Issue on Web Applications, July-August 1999.
4. Alfonso Fuggetta, Gian Pietro Picco and Giovanni Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, vol. 24(5), 1998.
5. Carlo Ghezzi and Giovanni Vigna, "Mobile code paradigms and technologies: A case study", Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, vol. 1219 of Lecture Notes on Computer Science, Springer, April 1997.
6. G. Glass, "ObjectSpace Voyager Core Package Technical Overview", Mobility: process, computers and agents, Addison-Wesley, Feb. 1999.
7. Dag Johansen, "Mobile Agent Applicability", Proceedings of the Mobile Agents 1998, Berlin, Springer-Verlag, Lecture notes in computer science; vol. 1477, ISBN 3-540-64959-X, (1998), pp. 9-11 September, 1998.
8. David Kotz and Robert S. Gray. "Mobile code: The future of the Internet", In Proceedings of the Workshop on Mobile Agents in the Context of Competition and Cooperation (MAC3) at Autonomous Agents '99, pages 6-12, Seattle, Washington, USA, May 1999.
9. D.Lange and M. Oshima, "Mobile Agents with Java: The Aglet API" In Special issue on Distributed World Wide Web Processing: Applications and Techniques of Web Agents, Baltzer Science Publishers, 1998.
10. Danny B. Lange and Mitsuru Oshima, "Seven Good Reasons for Mobile Agents", Communications of ACM, vol. 42, no. 3, March 1999.
11. Pattie Maes, Robert H. Guttman and Alexandros G. Moukas, "Agents That Buy and Sell", Communications of ACM, vol. 42, no. 3, pp. 81 - 91, March 1999.
12. Stavros Papastavrou, George Samaras and Evaggelia Pitoura, "Mobile Agents for WWW Distributed Database Access", Proceedings of IEEE International Conference on Data Engineering (ICDE99), 1999.
13. Gian Pietro Picco and Mario Baldi, "Evaluating Tradeoffs of Mobile Code Design Paradigms in Network Management Applications", Proceedings of 20th International Conference on Software Engineering, ICSE98, Kyoto, Japan, IEEE CS Press, 1998.
14. David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, Bill Peet, "Concordia: An infrastructure for collaborating mobile agents", In K. Rothermel & R. Popescu-Zeletin (Eds.), Lecture Notes in Computer Science: 1219, Mobile Agents, Proceedings of the First International Workshop (pp. 86-97), Berlin, Springer-Verlag, 1997.