Proceedings of the 2nd Asian International Mobile Computing Conference(AMOC 2002)
14-17 May, 2002
Langkawi, Malaysia

Performance comparison of implementation mechanisms for e-commerce applications: Towards a hybrid approach

Rahul Jha, Sridhar Iyer

KR School of Information Technology,
IIT Bombay, India, 400 076
{rahul,sri}@it.iitb.ac.in

Abstract:

The acceptance of Internet-scale systems and applications depends significantly upon their performance in terms of user-perceived quality of service. While there has been work in the domain of mobile agent based applications, there is little work to compare the performance of these implementations with client-server approaches. Also, there is a need to identify the application parameters that influence overall performance.

We have identified several parameters that influence performance and have carried out a quantitative evaluation of client-server versus mobile agent implementations. From our experiments, we conclude that both solutions need to co-exist. We then propose a *hybrid approach* using both these implementation strategies, and present the design of "MAX", an implementation of an e-commerce application. MAX is based on a hybrid MA/CS approach and chooses an appropriate implementation strategy at runtime, based on the application parameters.

1. Introduction

As the Internet evolves from an information space to a market space, electronic commerce is becoming an important mechanism for conducting business. This has created a need for new ways of structuring applications to provide cost-effective and scalable models.

Most existing implementations for structuring online market places use the traditional client-server (CS) design. Although extensively used, CS implementations have a non-intuitive conceptual model for applications involving mobility. The mapping of marketing actions to a request/response model, results in drawbacks such as increased network traffic, long duration connections and interaction, support for only standard queries and little support for disconnected operations.

The Mobile Agent (MA) design paradigm has been used to overcome some of these drawbacks [2,3,15]. A mobile agent is a program that can autonomously migrate between various nodes of a network and perform computations on behalf of a user. Designing MA based e-commerce applications is an effective conceptualization of traditional marketing, as it incorporates buyer's mobility and the notion of agents. MA supports execution of client specific queries, real time interaction, disconnected operations, reduced network traffic and faster response time in an e-market place.

While there has been work in designing MA based applications, the important issue of their performance versus traditional CS implementations, has received little attention. Numerous studies show that the acceptance of Internet-scale systems and applications depends upon their performance in terms of user-perceived Quality of Service [9]. This paper contributes towards identifying the application parameters that influence the performance (and thereby the design choices) for an e-commerce application. Our experiments are an effort towards arriving at guidelines for "*when to use MAs*" for a given application.

In the first part of this paper, we discuss some strategies for implementing e-commerce applications, using the CS and MA paradigms. We have carried out a quantitative performance evaluation of these strategies and have identified *performance crossover points*, viz. points at which any one strategy begins to perform better than another. We have used Java for the CS implementation and the Voyager framework [7] for the MA implementation.

From our experiments, we believe that there is no *"one-size-fits-all"* solution. Both CS and MA approaches have their own advantages, as well as scenarios under which one performs better than the other. We argue that the overall performance of an e-commerce application is not only dependent on the design choice (MA/CS) but also on application specific parameters. We claim that a *hybrid model* that uses both CS and MA in a complementary fashion may yield better performance than either CS or MA in isolation.

In the second part of this paper, we describe our implementation of such a *hybrid MA/CS model,* "MAX". MAX employs both the MA and CS implementations and *chooses one of them at runtime,* depending upon the values

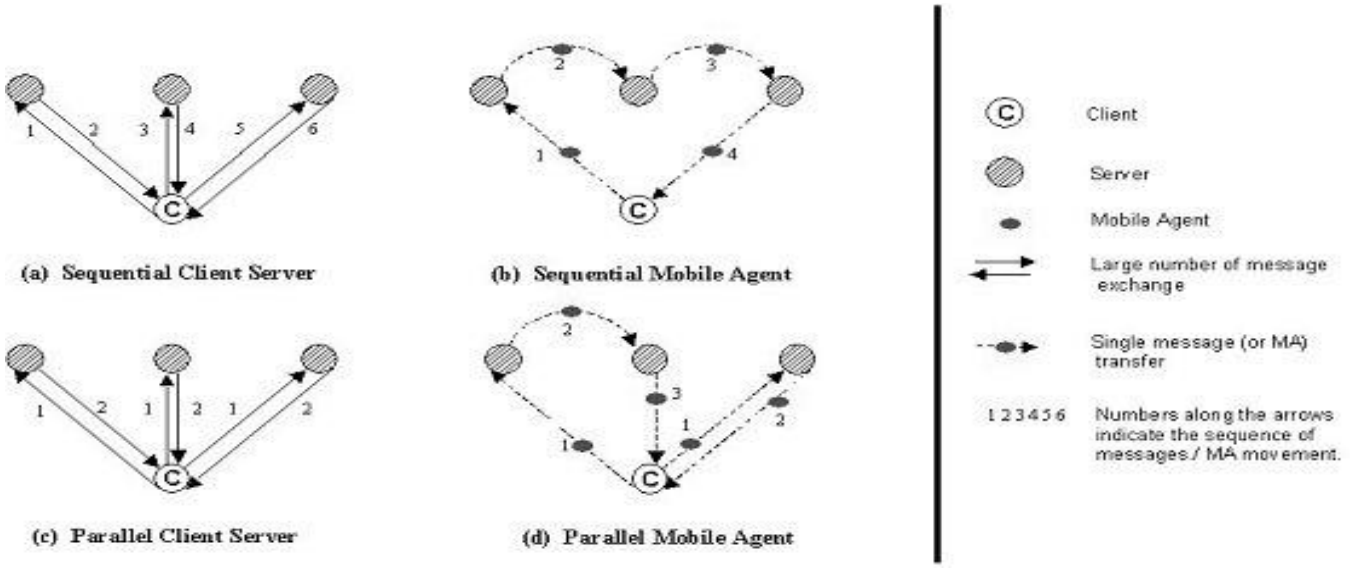

Fig 1: Implementation strategies

of the application specific parameters.

Section 2 discusses some strategies for implementing e-commerce applications, and Section 3 identifies some important parameters that affect their performance. Section 4 describes our experiments, and Section 5 the observations that lead to a hybrid model. Section 6 presents our hybrid model, and Section 7 the conclusions.

2. Implementation strategies

We identify four implementation strategies that may be adopted by a typical e-commerce application using the CS and MA design paradigms:

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. The selection of an "ideal" implementation strategy from those feasible for a given application, would depend upon application specific parameters. The following section identifies some application parameters that would play a crucial role in the overall performance, thereby influencing choice of the implementation strategy.

3. Characterizing application parameters

The performance of any distributed application depends upon the computation time at the client and the server, the network delays during data transfer, and the queuing/wait time at client/server. These delays depend upon the implementation strategy, which in turn is affected by the application parameters. We have identified the following parameters that influence the choice of an implementation strategy, for e-commerce applications:

- **Mobility patterns:** The shopping itinerary (the sites to be visited) is an important parameter that determines the *mobility pattern* of the execution. The itinerary could be either *static* (fixed at start of execution) or *dynamic* (changes during execution). The sites in the itinerary may need to be visited in a

given order or in *any order*, thereby imposing constraints on the suitability of an implementation strategy.

- **Catalog size:** Catalog size is the amount of data that is filtered at a server and sent to the client for processing. Non-standard queries and/or large server databases may lead to large catalogs being transferred over the network, thereby affecting the choice of an implementation strategy. CS implementations may not be suitable for large catalog sizes.
- **Number of shops:** The number of shops in the itinerary affects the overall data transfer and processing times. When coupled with large catalog sizes, this has a major impact on the implementation strategy. As the number of shops increases, parallel implementations may be preferred over sequential ones.
- **Network latency:** The network latency for an itinerary plays an important role in the choice of an implementation strategy. For low bandwidth or disconnected operations scenarios, MA implementations may be more suitable compared to CS.
- **Processing time:** For high bandwidth scenarios, the processing time becomes an important factor in the choice of an implementation strategy. CS implementations may be more suitable compared to MA, for some of these cases.
- **Product dependencies:** Dependencies among products to be discovered may impose additional constraints on the suitability of an implementation strategy. Parallel implementations may not be suitable when there are many such dependencies.

In following section we present our experiments to determine the performance of the implementation strategies discussed earlier, for various values of the above parameters.

4. Experimentation and results

We have chosen a typical e-commerce application of *product discovery*, viz., that of a single client searching for information about a particular product from the catalogs of several on-line shops. We assume that the client requires a highly customized query, which is not supported by the standard query interface of the on-line shop. Such a query would require the client to fetch a relevant subset of the catalog and implement a search at its end. We have implemented such an application using all the four strategies discussed in section 3.

4.1 Experimental setup

The experiments were carried out on P-III, 450 MHz workstations connected through a 10 Mbps LAN with typical student load. We have considered the following parameters for comparing the performance of these implementation strategies:

- mobility pattern (a static itinerary of 26 shops)
- catalog size (varies from 100 KB to 1 MB);
- number of shops (varies from 1 to 26);
- size of client-server messages (varies proportionately with catalog size);
- size of an MA (fixed at 4.6 KB);
- network latency (as per typical academic loads on 10Mbps LAN);
- processing time for servicing each request (varies from 20 ms to 1000 ms);

We have used the Voyager framework for MA (parallel and sequential) implementations [16]. Voyager is an ORB (Object Request Broker) implemented in Java and provides support for mobile objects and autonomous MAs [7]. 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 in Java.

Our performance metric is the *user turnaround time*, which is 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.

4.2 Evaluation of implementation strategies

We have performed experiments to determine:

- **Effect of catalog size on turnaround time for CS and MA**
  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, and the results are shown in figure 2.

- **Effect of server processing delay on turnaround time**
  The catalog size was kept constant at 1MB and server delays of 20ms, 500ms and 1000ms were considered. The user turnaround time was measured for different scenarios of product discovery from 1 to 26 shops. The different results are shown in the graphs of figures 3, 4 and 5.

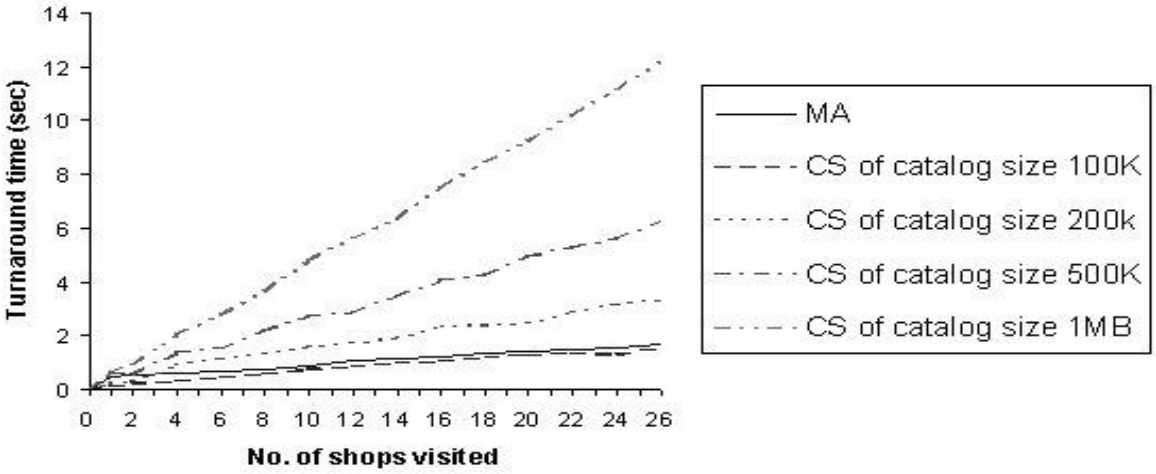

Fig 2: Effect of catalog size on turnaround time for sequential MA & sequential CS

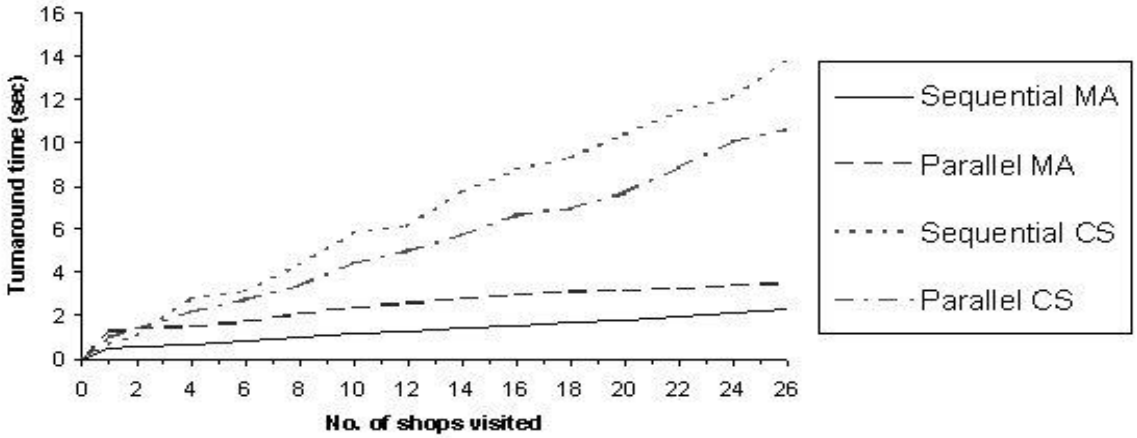

Fig 3: Turnaround time for a processing delay of 20 ms (catalog size of 1 MB).

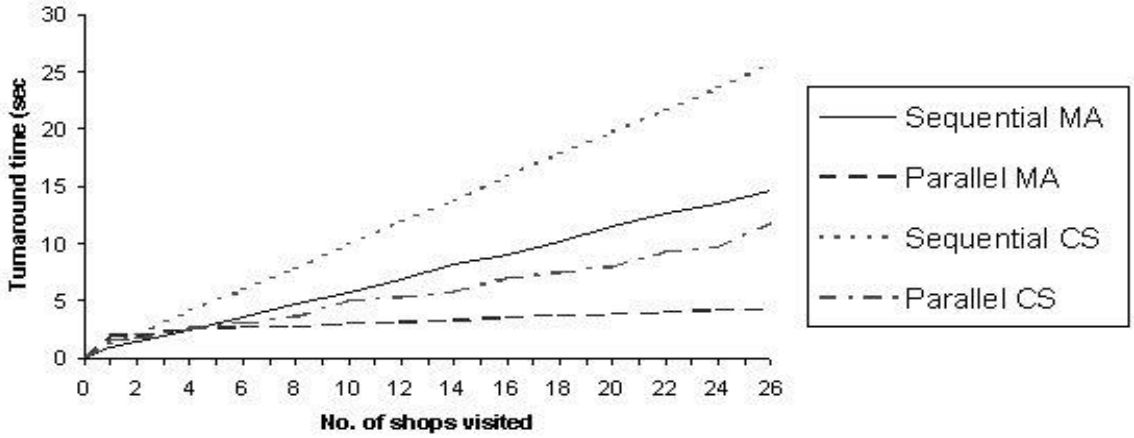

Fig 4: Turnaround time for a processing delay of 500 ms (catalog size of 1 MB).

Fig 5: Turnaround time for a processing delay of 1000 ms (catalog size of 1 MB).

4.3 Observations

The results of our performance evaluation experiments are shown in the graphs of Fig 2 to 5. 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. 2).
- CS implementations perform better than MA implementations for catalog sizes less than 100 KB (Fig. 2).
- 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. 2).
- MA performs better than all other strategies, for small processing delays (20 ms) and large (1MB) catalog size (Fig. 3).
- 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. 4).
- Parallel MA performs better than parallel CS for higher processing delays (1000ms) and large (1MB) catalog size (Fig. 5).
- *Performance crossover points* i.e. parameter values for which one implementation starts performing better than another implementation, for other combinations of application parameters, may be determined by inspecting the graphs in Fig 2-5.

5. Motivation for a hybrid model

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, when a "large" amount of information (greater than 500KB) is retrieved and for "large" number of servers (more than 6), MA implementations are more effective. The MA approach scales well as the size of data to be processed and the number of servers to be visited increases. Parallel implementations are effective when the processing delay contributes "significantly" (greater than 1000ms) to the turnaround time.

From our experience, we believe that both CS and MA solutions as well as sequential and parallel implementations need to co-exist. Neither a pure CS nor a pure MA approach is effective for all values of the application parameters. For the same application (product discovery), sometimes CS may be effective (less than 200KB of catalog size), while at other times MA may be effective (greater than 200KB of catalog size).

Since the values of these parameters may depend upon the type of product, the shop visited etc., they cannot be fixed a priori. Hence it is necessary to have a hybrid approach wherein the two paradigms (CS and MA) are used to complement each other. An ideal implementation would choose the most suitable approach, depending upon *runtime* values of the application parameters.

In the next section we describe the design and implementation of "MAX", our hybrid MA/CS model for e-commerce applications. Depending upon the application parameters at runtime, MAX uses the most suitable of MA or CS implementation strategies, for optimal user-level performance.

6. MAX: A MA/CS hybrid implementation approach

We have implemented "MAX" a complete customer driven, business-to-customer e-market place that supports

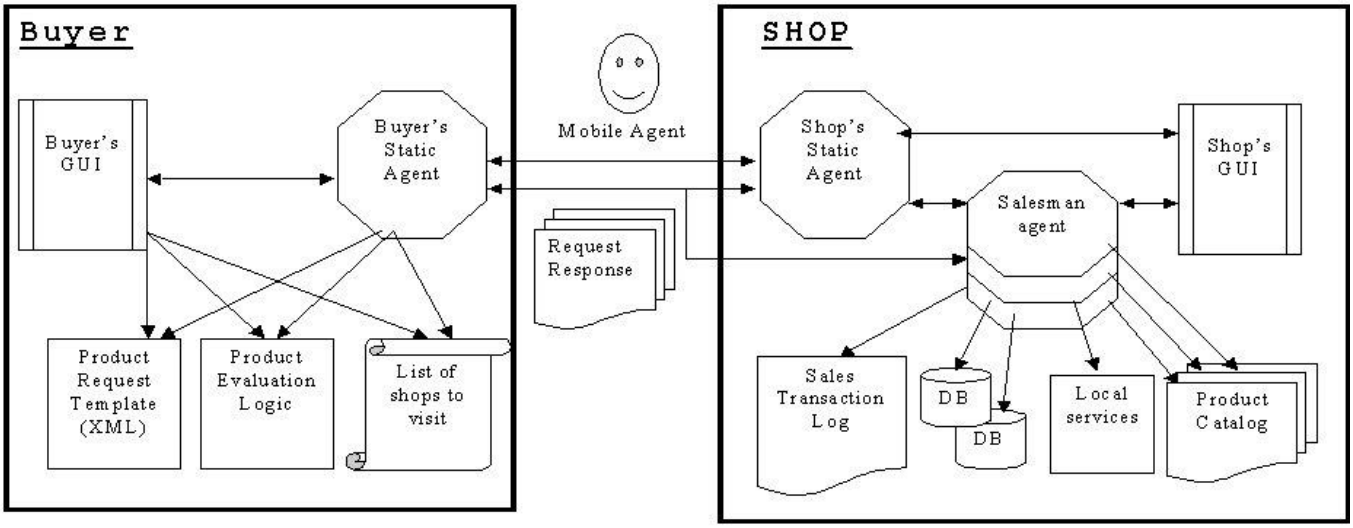

Fig 6: MAX: Architecture of hybrid e-commerce application

trading using both MA and CS implementations. Each shop in MAX has a static agent (SSA), which is capable of accepting both an MA as well as a CS request. Each client in MAX has a static agent (BSA), which is capable of launching mobile agents as well as initiating a client-server request-response communication.

The BSA chooses the implementation mechanism (MA/CS) at runtime, based on the parameter values. The observations of our performance evaluation experiments are used as training data and the performance crossover points as the guideline for choice of the appropriate strategy, based on the number of shops to be visited, and the estimated catalog size. The BSA also takes the network delay into account by performing a ping to one of the shops. If the available bandwidth is high, i.e., the ping round trip time is below a threshold, CS is chosen as the implementation strategy. If the available bandwidth is low, i.e., ping round trip time is above a threshold, MA is used to support asynchronous operations. For simplicity of implementation, these values are supplied by the user at the start of each product discovery. Although incorporation of learning and estimation algorithms into the BSA is a natural extension, these are beyond the scope of this paper.

6.1 MAX Architecture

The main entities in MAX are the buyer and shops. Each of these entities is composed of several components, as shown in Figure 6, and are described below:

6.1.1 Buyer side components

The buyer side implementation in MAX has the following components:

- **Buyer's Static Agent (BSA)**

The BSA is the local agent at the buyer's site and is responsible for managing local resources, services and agents. As described earlier, it dynamically chooses the implementation mechanism (MA/CS) at runtime. It interacts with the buyer to identify the list of shops, the nature of the product to be discovered, and creates a *product request template*. The user may also supply a *product evaluation logic* for rating products that match a request.

- **Product Request Template (PRT)**

The PRT is the representation of product parameters specified by the buyer. It is stored in an XML data format structure. For complex product requests involving sub-components, the BSA may split the PRT into smaller sub-graphs to enable parallel implementations.

- **Product Evaluation Logic (PEL)**

The PEL evaluates and rates products that match user's specification and appends chosen products to a *candidate product list*, which is returned to the user. The PEL provides for customized operations on the product catalogs, so that a buyer is not restricted to the standard query processing provided by a shop.

6.1.2 Shop's components

The shop side implementation in MAX has the following components:

- **Shop's Static Agent (SSA)**

The SSA is the local agent at the shopkeeper's site and handles all product purchase requests. It also maintains

the *product catalog* as an XML data format structure. The SSA is capable of accepting and servicing both client-server requests and mobile agents. It interacts with a *shop's salesman agent* to service each individual request.

- **Shop's Salesman Agent (SMA)**

Each SMA maintains the specific catalogs for a given product category. It handles all the information related to the product category such as logging of transactions etc. It provides the services such as filtering, searching, ordering, for each client's request.

6.2 MAX: MA implementation

The MA implementation in MAX is as follows:

1. The BSA creates a mobile agent with a *list of shops, the product request template and the product evaluation logic*.
2. At each shop, the MA interacts with the SSA. Based on MA's request, the SSA identifies a specific SMA for handling the request and informs the MA of the SMA.
3. The MA interacts with the specified SMA. The MA passes its product request template XML data tree to the SMA, which in turn does the product filtering from the product catalog XML data tree. The SMA passes the filtered catalog to the MA.
4. The MA uses its product evaluation logic to rate the filtered catalog supplied by the SMA and appends the chosen products to its candidate product list.
5. Upon completion of its itinerary, the MA returns to the BSA with the candidate product list and other information.

6.3 MAX: CS implementation

The CS implementation in MAX is as follows:

1. The list of shops to visit, product request template and the product evaluation logic are all retained in the BSA.
2. For each shop to be visited, the BSA creates a client request message and sends it to the appropriate SSA.
3. The SSA identifies an appropriate SMA for handling the request. The SMA carries out the necessary filtering and returns the filtered catalog.
4. The SSA returns the filtered catalog to the BSA in a server response message.
5. The BSA uses the product evaluation logic to rate the filtered catalog and appends chosen products to its candidate product list.
6. The BSA repeats steps 2-5 until all shops in the itinerary are visited.

We have implemented MAX in an experimental setup and are in the process of enhancing the application as well as carrying out further performance studies.

7. Conclusion

Performance of Internet-scale application being a critical factor for their acceptance, we have carried out experiments to compare the performance of CS versus MA implementations of an e-commerce application. We identified some application parameters, such as, mobility pattern, catalog size, number of shops, network latency, processing time etc. and found that the overall performance is not only dependent on the design choice (MA/CS) but also on the application parameters. Our experiments lead us to believe that a *"one-size-fits-all"* solution is unlikely to be effective.

We then proposed a hybrid approach that employs both MA and CS strategies and presented the design of "MAX", which is based on this hybrid approach. The use of a hybrid approach with the ability to choose the implementation strategy at runtime, makes it an attractive model for applications like e-commerce. We are in the process of incorporating learning and estimation algorithms into MAX and carrying out further performance studies.

ACKNOWLEDGMENTS

Authors are thankful to Vikram Jamwal and Srinath Perur for their suggestions and support during the development of MAX.

REFERENCES

[1] Jonathan Bredin, David Kotz and Daniela Rus, "Market-based Resource Control for Mobile Agents", Proceedings of Autonomous Agents, May 1998.

[2] P. Dasgupta, N. Narasimhan, L.E. Moser and P.M. Melliar-Smith, "A Supplier Driven Electronic Marketplace Using Mobile Agents", Proceedings of the First International Conference on Telecommunications and E-Commerce, Nashville, TN, Nov. 1998.

[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.

[7] G. Glass, "ObjectSpace Voyager Core Package Technical Overview ", Mobility: process, computers and agents, Addison-Wesley, Feb. 1999.

[8] Dag Johansen, "Mobile Agent Applicability ", Proceedings of the Workshop on Mobile Agents, Springer-Verlag, LNCS; vol. 1477, pp. 9-11 September 1998.

[9] B.Krishnamurthy and C.E. Wills. Analyzing factors that influence end-to-end Web performance. In Proceedings of the 9th International WWW conference, pp. 17-32, May 2000.

[10] 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, Seattle, USA, May 1999.

[11] Danny B. Lange and Mitsuru Oshima, "Seven Good Reasons for Mobile Agents ", Communications of ACM, vol. 42, no. 3, March 1999.

[12] 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.

[13] Stavros Papastavrou, George Samaras and Evaggelia Pitoura, "Mobile Agents for WWW Distributed Database Access ", Proceedings of IEEE International Conference on Data Engineering (ICDE99), 1999.

[14] G. Samaras, M. Dikaiakos, C. Spyrou, and A. Liverdos., "Mobile agent platforms for webdatabases: A qualitative and quantitative assessment.", In Proc. of the 1st Int'l Symp. on Agent Systems and Applications and Third Int'l Symp. on Mobile Agents (ASA/MA'99), pages 50--64. IEEE Computer Society Press, October 1999.

[15] Thomas Sandholm and Qianbo Huai, "Nomad: Mobile Agent System for an Internet-Based Auction House ", IEEE Internet Computing, vol. 4, no.2, March-April 2000.

[16] Rahul Jha and Sridhar Iyer, “Performance evaluation of mobile agents for e-commerce applications”, International Conference on High Performance Computing, Hyderabad, India, Dec 2001.