Cluster Computing 5, 377–388, 2002

Homogeneous Agent-Based Distributed Information Filtering

RAJEEV R. RAJE, MINGYONG QIAO, SNEHASIS MUKHOPADHYAY, MATHEW PALAKAL and SHENGQUAN PENG

Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 723 W. Michigan Street, SL 280, Indianapolis, IN 46202, USA

JAVED MOSTAFA

School of Informatics, Indiana University, 1320 East tenth street, Room 025, Bloomington, IN 47405, USA

Abstract. Recent advances in processor, networking and software technologies have made distributed computing a reality in today's world. Distributed systems offer many advantages, ranging from a higher performance to the effective utilization of physically dispersed resources. Many diverse application domains can benefit by exploiting principles of distributed computing. Information filtering is one such application domain. In this article, we present a design of a homogeneous distributed multi-agent information filtering system, called D-SIFTER. D-SIFTER is based on the language-dependent model of Java RMI. The detailed design process and various experiments carried out using D-SIFTER are also described. The results indicate that the distributed inter-agent collaboration improves the overall filtering performance.

Keywords: multi-agent systems, information filtering, distributed-object computing, representation and clustering, Java-RMI

1. Introduction

In recent past, with the proliferation of personal computers, individual workstations and the advent of high-speed networks; distributed or clustered computing has become a prominent paradigm. There are many reasons for the popularity of distributed computing, e.g., speedup, inherent distributed properties of the application, a presence of specialized functional units and geographically dispersed data resources. The development of distributed software systems involves handling many challenging issues, such as heterogeneous computing resources, multiple languages, partial failures, communication, etc. Some of these issues can effectively be handled by the use of a platform independent language like Java and its distributed computing feature – Remote Method Invocation (RMI) [19]. In this paper, we describe Distributed Smart Information Filtering System for Electronic Resources (D-SIFTER) – a homogeneous agent-based distributed information filtering system – that is designed using the distributed-object model of Java-RMI.

The rest of this paper is organized as follows. We start with a brief introduction of the philosophy of distributed information filtering. It is followed by a discussion about related and our past work. Then we describe the detailed design and implementation of D-SIFTER. Finally, we present a few experiments, conducted with D-SIFTER, along with the results and conclusions.

2. Related and previous work

2.1. Information filtering and retrieval

In an interconnected world, a tremendous amount of electronic information is created and delivered each day. The flooding of the information over the network has forced users to locate tools that will assist in coping with this information overload. Information Filtering (IF) and Information Retrieval (IR) systems are examples of such tools.

IF is a way to classify, sort and present documents according to an user's interests. IF systems are commonly personalized to support long-term information needs of a particular user or a group of users with similar needs. They accomplish the goal of personalization by directly or indirectly acquiring information from the user. In IF systems, these long-term information needs are represented as user interest profiles [5], which are subsequently used for matching or ranking purposes. The interest profiles are maintained beyond a single session and are modified based on users' feedback. As user profiles are indications of users' behavior, it is necessary to protect the privacy of these profiles. In this information age, the importance of IF systems is paramount. They aid in reducing the information burden of users and at the same time make an attempt to adapt to the changing user interests.

Filtering contexts typically involve a dynamic stream of information, as opposed to static data bases used in traditional IR systems. Due to the dynamic nature of the stream, timeliness of information assumes a added significance in the filtering context. The amount of data involved in filtering environments is usually very large and unstructured. IF involves repeated interactions over multiple sessions with users having long term goals. This is in contrast to IR systems, where the user typically has a short term information need that is satisfied within a single session.

Filtering systems are more likely to be used by a wider community of users than IR systems. The users of IF systems may not be highly motivated in their information seeking process and may not have well defined interests. Therefore, IF systems need to be more user-friendly than IR systems, which

have been mostly aimed at the highly motivated information seeker with very specific information needs.

2.2. IR systems

IR is a well established field of information science that addresses issues of retrieval from a large collection of documents in response to user queries. Wide Area Information Servers (WAIS) [3] is a networked-based document indexing and retrieval system for textual data. In WAIS, servers maintain inverted indices of keywords that are used for efficient retrieval of documents. WAIS allows users to provide relevance feedback to further specialize an initial query. Gopher [6] is primarily a tool for browsing through hierarchically organized documents, but it also allows to search for information using full-text indices. In the World Wide Web (WWW) [1] the information is organized using the hypertext paradigm where users can explore information by selecting hypertext links to other information. Documents also contain indices which the user can search for.

2.3. Agent-based information services and IF systems

Amalthea is a multi-agent evolutionary system based on genetic algorithms applied to IR [8]. In this system, agents perform tasks such as: learning user interests, discovery of relevant information, information filtering, and monitoring of periodical content changes of specific sources. There are two types of agents in the system, information filtering agents and information discovery agents. These two agents compete and cooperate to serve the user.

Stanford Information Filtering Tool (SIFT) maintains user profiles, which are composed of keyword inputs by the user, compares the incoming news against the profiles and email the relevant news to the user [21].

Pefna [4] system filters newsgroup articles using representative articles for different categories. The user provides a list of ordered categories and representative article for each category. The articles are rated by assigning a cosine distance value and comparing them against the sample article for every category. Articles below a certain threshold are then eliminated.

D-SIFTER has some similarities and a few significant differences with the above mentioned systems. These similarities and differences will become more apparent as we discuss the philosophy and design of D-SIFTER.

3. Previous work

3.1. SIFTER

D-SIFTER has its roots in SIFTER, which is a single agent and stand-alone information filtering system, developed at IUPUI (Indiana University Purdue University) [7]. SIFTER consists of four modules: *a representer, a classifier, a user profiler*, and *a presenter*. The representation module is responsible for converting a document into a numeric structure that can be manipulated by the classification module. The classification module consists of two important stages: the cluster learning stage and the vector classification stage. During the first stage, clusters are generated from an initial set of sample documents. Each cluster is then represented by its centroid. Each cluster and its corresponding centroid signifies a different document category. It is then the task of the classification module to classify each incoming document into a specific cluster. The purpose of the user profile module is to keep the user's profile, the mechanism by which the system views the human user, up-to-date. The presenter module allows interactions with the user. The combination of these four components constitute a filter or an agent, which performs the task of rank-ordering and presenting the incoming documents based on the user's preferences.

In SIFTER, a filter (or an agent – in this article, the terms agent and a filter are used interchangeably) contains a knowledge base, called *thesaurus*, that consists of keywords and terms culled from an authoritative source of a specific domain. The representer converts the incoming document into a vector based upon the thesaurus. As stated above, the classifier uses this vector to classify the document to a specific cluster. Due to the discovery of newer keywords as well as computational complexity resulting from a large thesaurus, it is possible that an agent's thesaurus cannot contain all possible keywords/terms. Such a limitation of the knowledge base results in many incoming documents not being classified and presented to the user, thereby, resulting in a poor information filtering performance.

3.2. Rationale for using distributed information filtering

The above mentioned and other inherent limitations of SIFTER can be elegantly handled by the use of distributed information filtering principles involving collaboration between multiple agents equipped with separate thesauri. Below, we present a brief discussion about the rationale for D-SIFTER. A more detailed elaboration is found in [11].

SIFTER suffers from the inadequacy of its thesaurus. Two alternatives exist to overcome this drawback. The first one aims at making the thesaurus extra large; while the second alternative uses the principles of distributed systems by allowing multiple agents to collaborate with each other over the network. Each agent in the multi-agent system has a small thesaurus that is specific to one domain (i.e., Computer Science, Biomedical Science, etc.), or catering to an user's interests (i.e., books, music, sports). In case the thesaurus of an agent turns out to be inadequate, that agent can seek assistance from other agents.

The first approach has distinct disadvantages such as, time complexity, difficulty in dealing with independent discovery of new terms, poor scalability, fault tolerance, adaptability, and flexibility. The second approach easily overcomes these drawbacks. Multiple agents cooperating with each other speed up the computation, complement each other's capabilities, share each other's knowledge, and improve the efficiency and quality of information filtering.

3.2.1. Speedup and time complexity

As the number of terms in a thesaurus increases, more centroids are generated during the cluster learning stage. We performed various experiments [11] with different thesaurus sizes (ranging from 10 to 80) on 5000 documents. As expected, the number of centroids generated increased almost linearly with the size of thesaurus. During the process of classification the distances between the incoming document vector and all centroids are computed and the the incoming document is assigned to the centroid having the minimum distance. Thus, it is obvious that larger the number of centroids, more time would be needed by the classification process; thereby, increasing the overall filtering time. Hence, a very large centralized thesaurus would certainly suffer from poor response time. In contrast, a multi-agent filtering system will consist of many agents, each having a small thesaurus, thereby, reducing the classification time and improving the overall response time. This improvement may be possible in spite of the fact that the multi-agent approach has to deal with the network transmission overhead.

3.2.2. Fault tolerance

In a centralized system, everything including thesaurus, centroids, classified documents, user profiles, are all located on one machine. If that machine fails due to any reason then users cannot get information filtering services. In a distributed information filtering environment, if one machine fails, users can get a reduced performance with a graceful degradation by employing other agents executing on different machines.

3.2.3. Flexibility and adaptability

In a centralized system there is only one thesaurus. The centroids and the user profiles are specific to that thesaurus. As new document streams may arrive and/or user's interests may change, the thesaurus may not be good enough to represent the document contents or new interest domains. Hence, there will be a need for updating the thesaurus. Our experiments with expert users from biomedical domains have indicated the need for automatic discovery of new terms and a constant upgrading of the thesaurus. If the thesaurus is updated, the centroids and user profiles need to be updated also. This obviously is not an easy task, especially in a centralized scenario. In the centralized model, one way is to handle the updating task is to re-learn the clusters and user profiles, which is a time consuming process. In a distributed IF system, this problem can be solved easily. New agents with new thesauri and centroids can be added to the system without changing the existing agents. User can always change interests and get documents classified from other agents. A distributed information filtering system can also have multiple agents, each serving a different domain and/or a user community. Thus, a distributed approach to information filtering exhibits openness.

3.2.4. Resource sharing, privacy and economics

In real world, users and data resources are geographically dispersed. Users may need to get information from different databases and databases may want to serve different users. An user wants to quickly get the information that he/she is interested in. Since documents are located in different places and stored in different formats, there is not an easy way to put all information onto one machine due to copy rights, privacy, etc. Companies and organizations may want to earn money by giving the information which users want. Users may only want to buy the information that they are interested in. In such a case, a distributed information filtering system is obviously appropriate, as: a) there could be local filtering agents for each data resource, b) different agents may collaborate through adequate economical models and policies, and c) an user's agent can get remote documents by paying appropriate compensations. In addition, users may not want to put their profiles on this centralized machine, as it would compromise their privacy. In turn, they would prefer maintaining their profiles on their local machines. Such a distribution of profiles is clearly infeasible in the centralized model.

D-SIFTER precisely addresses these limitation of the centralized approach by exploiting the interconnected nature of a net-centric environment. D-SIFTER adds a distributed flavor to SIFTER by creating a multi-agent environment, in which an agent, in addition to serving its user, aids other agents or asks for assistance from others. D-SIFTER is a homogeneous IF system, as all agents in D-SIFTER are identical except for their thesauri. Each agent has a small thesaurus, which may or may not have overlap with the thesauri of other agents. Whenever an agent cannot classify an incoming document, due to its limited thesaurus, it seeks assistance from other agents. Depending upon the collaboration model used (details are mentioned in section 4), an agent may receive assistance from one or many other agents. Such a inter-agent collaboration not only increases the classification but also improves the overall filtering performance.

4. D-SIFTER

We have designed D-SIFTER using the distributed-object model of Java-RMI. Hence, before we discuss D-SIFTER, in detail, below a brief review of Java-RMI is provided. For more details the user is referred to [10,12,19].

4.1. Java RMI

The basic idea behind Java RMI is the concept of a remote object. A remote object supports invocations of its remote methods by other objects regardless of their address spaces.

When a client object communicates with a remote server object, which resides in a separate Virtual Machine, it (client) must interact with the remote interface of the server. The remote interface defines the methods of the server that are visible to nonlocal objects. The client receives a reference to the server through a naming service, called *Registry*.

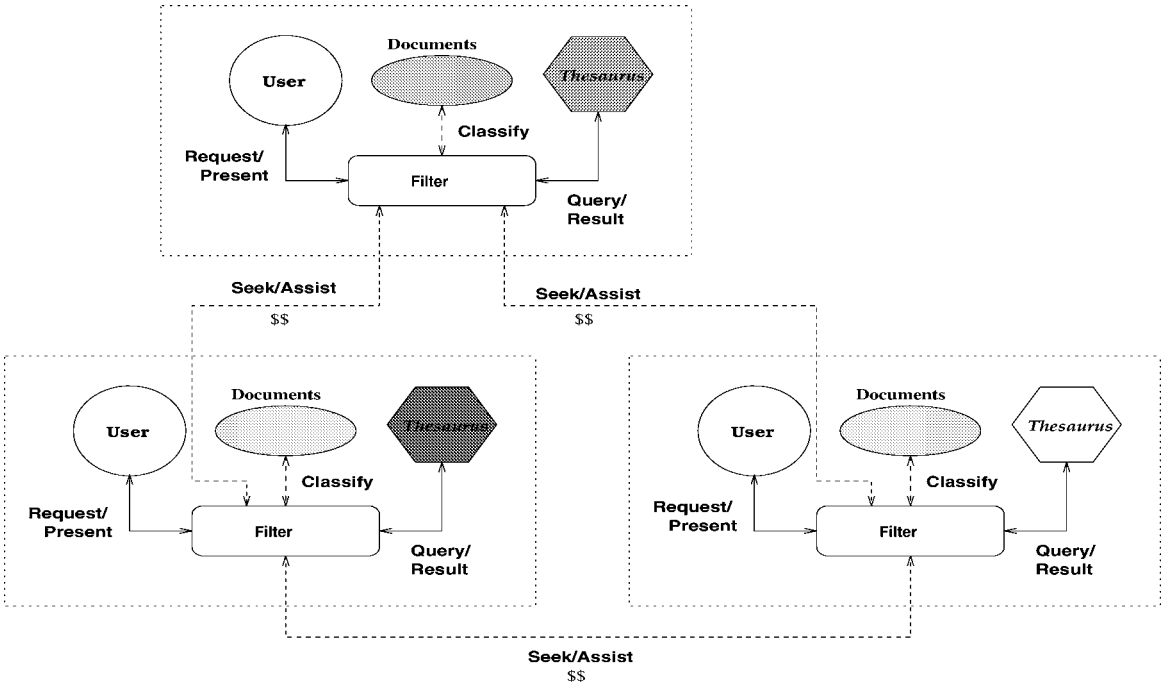

Figure 1. Distributed information classifier.

When a client invokes a method on a remote interface, a proxy object, called the stub, receives the method invocation. The stub then forwards the call across the network, where a skeleton receives it. The skeleton will then invoke the method on the server object. When the invocation is done, any return arguments are forwarded back across the network to client object through the same mechanism.

We chose Java RMI as the preferred model of the implementation of D-SIFTER due to its many attractive features. The homogeneous nature of the agents in D-SIFTER mapped well with the language-centric model of RMI. The simplicity of Java RMI was another reason for its selection. Other inherent features of Java, such as portability, built-in security and the support for the applets were additional reasons for making RMI as the preferred model for the design of D-SIFTER.

4.2. Distributed information classifier

The distributed information classifier (DIC) [14] is the first part of D-SIFTER. It focuses only on inter-agent collaboration for the purpose of document classification. In the distributed classifier, we have investigated many paradigms for inter-agent collaborations. In addition to the representer, classifier and the user interface modules, which are identical to the ones in SIFTER, each agent is equipped with a collaboration module. This module handles the inter-agent communications.

Collaborative Paradigms. Figure 1 shows the architecture of DIC. As mentioned earlier, in DIC all agents are identical, except for the thesaurus. The communication among different agents takes place in an indirect manner through a common server. The server architecture is fairly simple – it has a waiting queue for storing the documents that need assistance from other agents. Each agent has its own result queue for storing the results of the classification. When an agent fails to classify a document, it puts that document into the waiting queue on the server. The agent periodically checks its result queue to see if there are any classification results, made available by other helping agents, which need to be retrieved.

In DIC, we have implemented three types of collaborative schemes: single-opinion, multiple-opinion and combined-opinion.

- Single-opinion: In this model, only one remote agent is allowed to assist another agent. The strength of this model is simplicity, but it also results in the drawback of decreasing the possibility of all documents being classified.
- Multiple-opinion: In this model, more than one remote agent is allowed to assist another agent. It aims to compensate the drawbacks of the single-opinion model. It represents the real world by allowing different agents to participate in the collaboration. It leads to a better classification. The drawback of this model is the difficulty in managing multiple remote agents. The critical issues include waiting for all agents to complete their classification, waiting for all agents to produce a bid (in a market driven system), and an added overhead of determining the most appropriate remote classification of a document.
- Combined-opinion: This model aims at overcoming the drawback of the single-opinion model, and at the same time, simplify the complexity of the multiple-opinion model. It allows all remote agents a shot at classifying a document, that cannot be handled by the local agent, but in a sequential order. Thus, if an agent, a_i, cannot classify a document, d_{ik}, then all other agents, a_j, $i \neq j$, attempt to classify d_{ik} in a sequential manner. As soon as any agent,

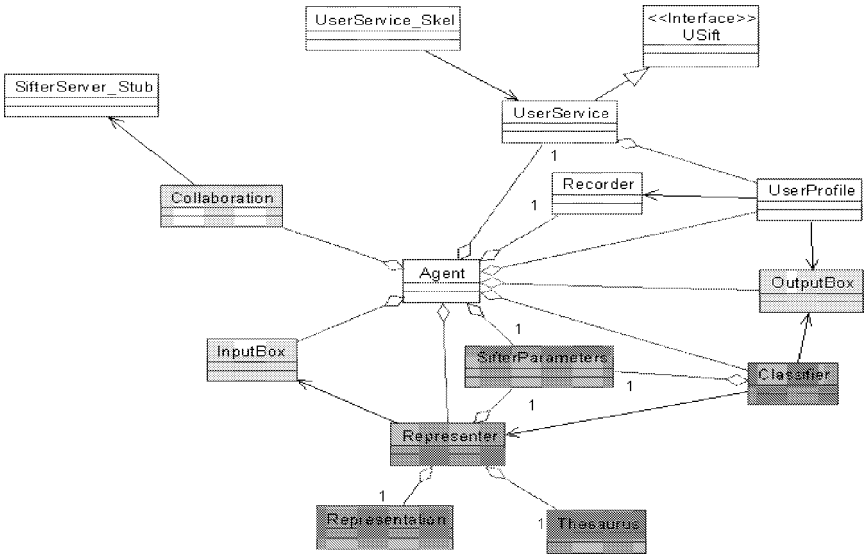

Figure 2. Logical view of an agent.

a_k, is able to classify d_{ik}, the remaining agents are denied the chance of classifying d_{ik}. This model guarantees that a document will be classified if there is at least one agent which has an adequate thesaurus to classify that document.

Below, we present the design of D-SIFTER that contains, in addition to DIC, the distributed profile module. Thus, an agent in D-SIFTER contains a representer, a classifier, a collaborator, a profiler and a presenter. Again, all agents are identical in structure, except for their thesauri.

5. D-SIFTER design with the distributed profile module

5.1. Architecture

In D-SIFTER, the design of an agent is divided into three packages: SIFTER package, communication package, and user service package. SIFTER package itself is a complete and stand-alone information filtering agent. Communication package is responsible for collaboration with other agents. User service package consists of an applet and several other modules, which provide the user with an interface for interactions with the system. In addition to the agents, D-SIFTER also contains a module, *SiftServer*, which is the common server. This common server allows inter-agent communication in an indirect manner. There are two reasons for not incorporating a direct multicast in D-SIFTER. The first reason is simplicity of the design. The second reason is that in a market-based collaboration, a presence of a central authority helps in the regulation of the system dynamics. The logical view of the D-SIFTER, using the UML notations, is shown in figures 2 and 3.

5.1.1. SIFTER package

This package includes the Representation module, Thesaurus module, Classifier module, User Profile module, InputBox module and OutputBox module.

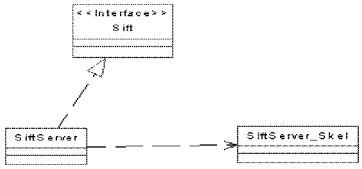

Figure 3. Logical view of SiftServer.

Representation module. This module is responsible for converting a document into a numeric structure that can be manipulated by the classification module. To accomplish this SIFTER uses the common vector-space model [18] for the representation and the popular *tf-idf* technique to establish a degree of importance for each term in that document. This module creates a table for each document that contains the frequencies of thesaurus terms in that document. Then, the following equation is used to calculate the elements of the vector: $W_{ik} = T_{ik} \cdot \log(N/n_k)$, where T_{ik} is the number of occurrences of term T_k in the current document, i, $\log(N/n_k)$ is the inverse document frequency of term T_k in the entire document set, N is the total number of documents set, and n_k is the number of documents in the set that contain the given term T_k.

Thesaurus module. This module represents the knowledge base for a particular domain. If we want to change the domain of the documents or expand the knowledge base of the agent, we simply change the contents of this module. Such a design provides the necessary flexibility to adapt D-SIFTER to multiple domains, with the minimum amount of modification.

Classifier module. This module consists of two important stages: the cluster learning stage and the vector classification stage. During the first stage, clusters are generated from an initial set of sample documents vectors. Each cluster is then represented by its centroid. Each cluster and corresponding centroid signify a different document category. The simple

Maximum-Distance algorithm [20] is used to determine the centroids over the document vector space. It is then the task of the classifier module to classify each document into a specific cluster. We used the distance between the documents and the centroids to determine which cluster the document belongs to. The cosine similarity measure [17] is used to compute this distance.

User Profile module. This module maintains the user's profile. The function of the user profile is to dynamically maintain a record of the user's preferred document classes through the feedbacks, and to prioritize the incoming documents. The algorithm used to learn the user model is based on a reinforcement learning algorithm from the area of Learning Automata [7,9] in AI and Mathematical Psychology communities.

InputBox module. This module provides a wrapper to the incoming the documents, so that all documents will have an unified form, irrespective of their origin – local or remote. The advantage of a wrapper is that the SIFTER package does not need know anything about the external environments.

OutputBox module. This module is used to manage the classification results. The advantage of having a module to manage the results is that the results can be reused. Thus, we can always have a thread to classify the documents if there is a document in the InputBox.

5.1.2. Communication package

The communication package consists of two modules: a Collaboration module, which is responsible for communicating with other agents in the system and a Recorder module, which is responsible for implementing the distributed user profile.

Collaboration module. This module, which is a client of the SiftServer, is used to expand the single agent SIFTER to a multiple agent system. When there is an unclassified document, the agent will sent that document to the SiftServer via the Collaboration module. The agent will also check if there is a result that needs to be brought back from the SiftServer. Currently, the communication is implemented in Java RMI. If we want to change the implementation to another mechanism, we can simply change this module and that change will not effect other parts of the system.

Recorder. This module is used to implement the distributed user profile. As stated earlier, each incoming document is assigned to a cluster, by the classifier. All the clusters (which are created off-line using a sample document set) are identified by unique numbers. Each agent, at any time, keeps the track of total number of classes known to itself. This total number consists of the number of the local classes and the number of remote classes created as a result of the inter-agent collaboration.

The function of the recorder module is to dynamically maintain a map from local class numbers to the unified class identifiers. At the beginning, the map has only information about the local classes. When an agent gets a result back from a remote agent, as a result of a collaboration, it will check the remote agent identifier and the class number, as indicated by the remote agent. If the combination of the remote agent id and the class number is not present in the local map, the local agent increases the total class number by one and adds a new entry into its map. This entry is a tuple of the form $<agent - id, classno>$. Eventually, the map becomes stable and static, and no new entries are added into this map. When another new agent enters in the system, or some agents expand their knowledge bases, the map is again updated. If all of the agents are allowed to help, the map will consist of entries corresponding to all possible classes in the entire distributed system. Thus, if there are N agents in the system and if each agent, A_i $(1 \leqslant i \leqslant N)$, has c_i local classes then in the case of an agent receiving assistance from all others in the system, the maximum possible entries in the local maps will be $\sum_{i=1}^{N} c_i$. If only a part of the agents are allowed to help, then the map will contain a subset of total number of classes possible in the system.

In D-SIFTER, we implemented the map as a hash table. Figure 4 shows the sequence diagram used in establishing the map for each agent. The basic algorithm followed by each agent is:

1. If a document cannot be classified then:
   (a) Put the document into the waiting queue of the SiftServer by invoking *storeDoc(DocRequest)* function of the SiftServer. This invocation is a remote call using RMI semantics. The Collaboration module is responsible for storing the documents on the server.
   (b) Periodically, check if one (or more) of your previously submitted documents have been classified by a remote agent and the result is available on the server:
      (i) if such a result is available, then retrieve that result by invoking *getResult()* remote function of the SiftServer;
      (ii) place the result into its own OutputBox;
      (iii) check the tuple $<agent - id, classno>$ of the result fetched against entries in the class map;
      (iv) if the new tuple is not present in the class map then add the entry into the class map.
2. Repeat the process for all other documents, which cannot be classified locally.

Figure 5 shows the sequence diagram of updating user profile. When an user logs into the system, the User Service module will initialize an user profile object on behalf of the user. The user profile will check with the Recorder module and obtain the current total class number. If the total class number has increased, the user profile is expanded to this new size and new classes, which have been obtained as a result of remote collaborations, are incorporated. The detailed algorithm of expanding the user profile is presented in [16].

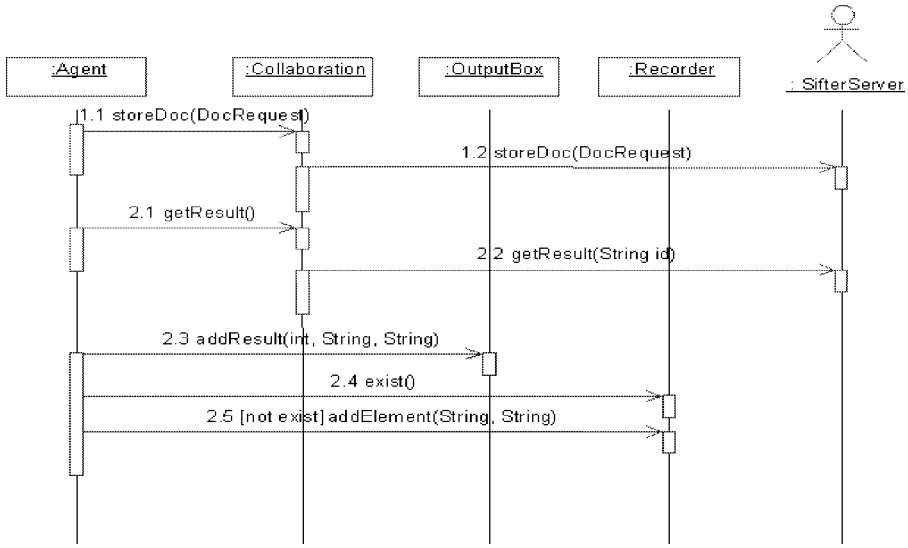

Figure 4. Sequence diagram for managing the Recorder table.

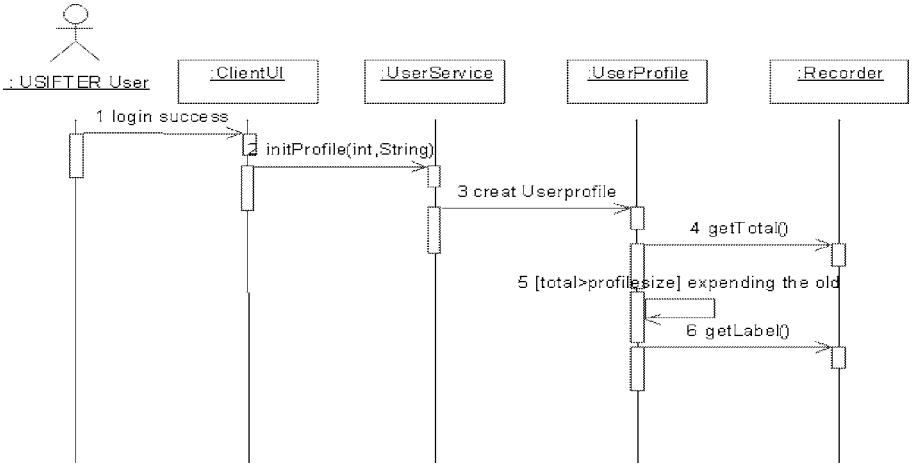

Figure 5. Sequence diagram for updating the User profile.

The user interface is implemented as a Java applet. In fact, this applet is a client of the D-SIFTER system. Whenever an user logs into the system, the system checks the user's ID and password. If it is a valid user, the main user interface pops up, from which the user can interact with the D-SIFTER. After the initial session, each further interaction of the user with D-SIFTER starts with an automatic presentation of the rank-ordered documents, available so far, to the user. There are many other menus available as a part of the user interface. The user can initialize the profile when he first logs into the system or when the user wants or change his interests dramatically. This avoids the extra effort and the overhead of establishing the profile from scratch. The user can also give the feedback to each document, so that the system can capture users' interests accurately. The system also provides a profile converge graph to help the user to determine if the system has totally learned the user's profile. Figure 6 shows the interface which displays the rank ordered titles of the documents and the associated menus. Each page displays 20 documents. The user can read more titles by pressing the more button. Figure 7 shows the interface which displays the entire text of a specific document. The user can provide a relevance feedback factor, from 0 to 1, for each document or simply close this window.

5.1.3. SiftServer

The communication among different agents takes place in an indirect manner through the common server. The current architecture of the server is fairly straightforward. It has a waiting queue for storing the incoming documents needing remote assistance and each agent has its own result queue for storing the results of the classification. The basic idea is that when an agent fails classifying a document, it will put this document into the waiting queue on the server. The agent will also periodically check its result queue to see if there are classification results to be brought back. This module is implemented as a Java RMI object, which has a published remote interface, so the agents can invoke remote functions on it and achieve the necessary communication with it. In RMI, a remote object is identified by a name and an associated URL-like id. The re-

mote object registers itself with the Registry by entering the URL-format id and a name. The agents, before invoking a remote function on the SiftServer, query the registry by providing the name (a string) of the server and in return, obtain the URL-format id of the SiftServer. Once this id is obtained, all remote calls are initiated through this id.

5.2. Multithread synchronization

D-SIFTER provides an ideal environment for incorporating multiple threads. Each user has different threads to handle the process of initializing the profile, retrieving and sorting the documents and collaborating with other agents. These threads must be synchronized very carefully as to avoid the race conditions and deadlocks. Also, the OutputBox and the SiftServer need a careful attention, as there are many threads accessing these objects at the same time. The SiftServer is designed as a monitor, in which all its methods are synchronized. Thus, during a concurrent access, only one thread will have the access to SiftServer. In a distributed system, there is a high probability of a failure. Each remote method of SiftServer throws a `RemoteException`, thereby, providing an exception handling capability.

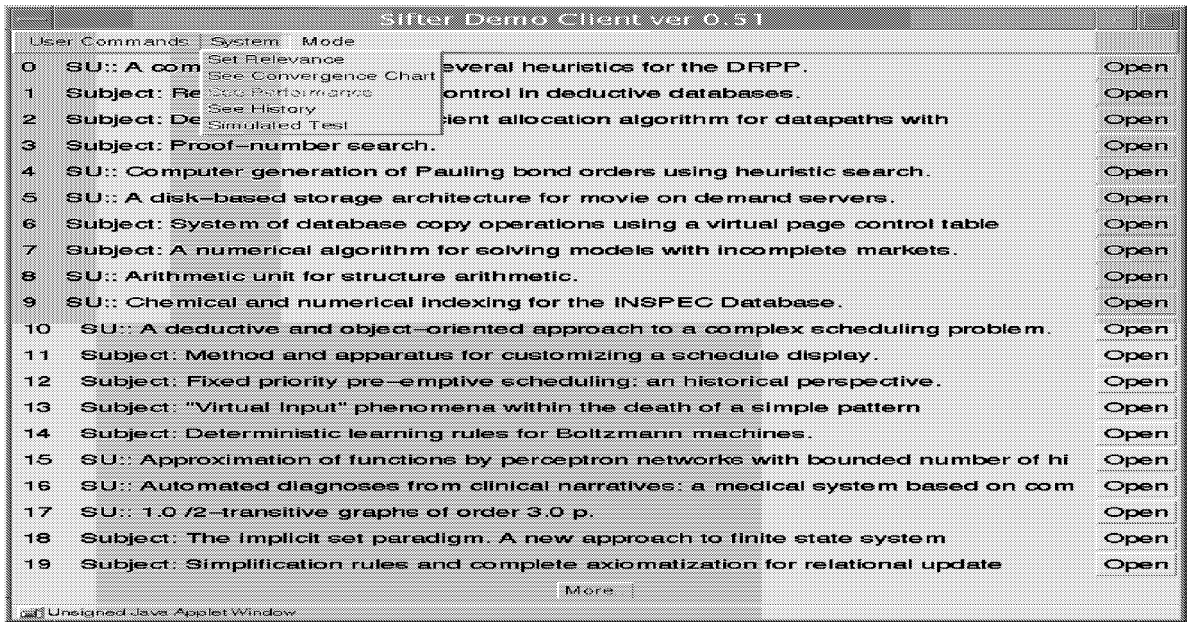

Figure 6. GUI of D-SIFTER: the interface which displays the rank ordered titles of the documents and the associated menus.

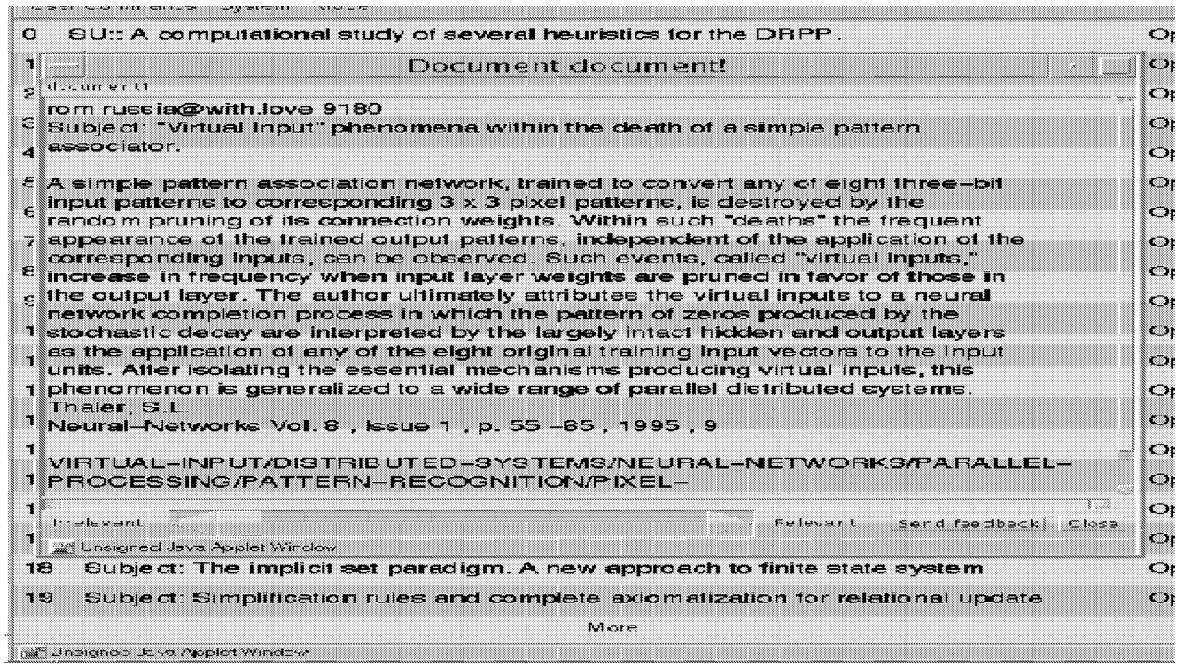

Figure 7. GUI of D-SIFTER: the interface which displays the entire text of a specific document.

6. Experiments and results

Although, we have performed many experiments with D-SIFTER, here, we present a few sample ones. More details are presented in [16].

6.1. Collaborative classification

We investigated the collaborative classification using combined-opinion model. The total number of incoming documents used for the experiment was 500. Initially only one agent (Agent2) was allowed to classify the incoming documents without any collaboration. It was able to classify 162 documents. Thus, its success ratio was 32.4%. Afterwards, this agent was assisted by other two agents each having a different thesaurus. The classification process using the same 500 documents was again carried out. As shown in table 1, Agent2 with the collaboration was able to classify 473 documents, which indicates the success ratio of 94.6%, which is a significant improvement over the stand-alone scenario.

Table 1
Number of successful classifications.

System	Agent1	Agent2	Agent3	Total	Success
Single-agent	X	162	X	162	32.4
Multiple-agent	176	162	135	473	94.6

6.2. Performance of D-SIFTER

Next we present the results of one experiment which indicates the improvement, as a result of collaboration, in the overall filtering performance. As running filtering experiments with the real users are costly, due to the human effort and time involved, in this test, we used a simulated user instead of a real user. In our previous work [7], we have proved that a strong correlation exists between the real and simulated users. Hence, although this experiment used a simulated user, it closely depicts the real user scenario. We created a simulated user, who was interested in the documents belonging to four different classes. One class was a local class and other three classes were remote classes. Relevance factors for these four classes were set to 1.0, and for others set to 0. We used two popular parameters, normalized precision and normalized recall [7], to evaluate the performance of D-SIFTER. Figures 8 and 9 indicate the results of this experiment. As seen from

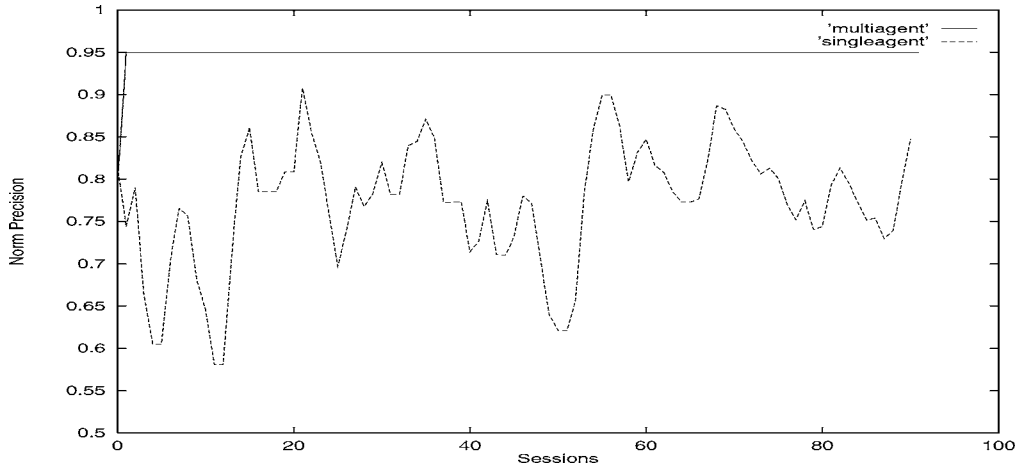

Figure 8. Comparison of the performance of Single and Multi agent.

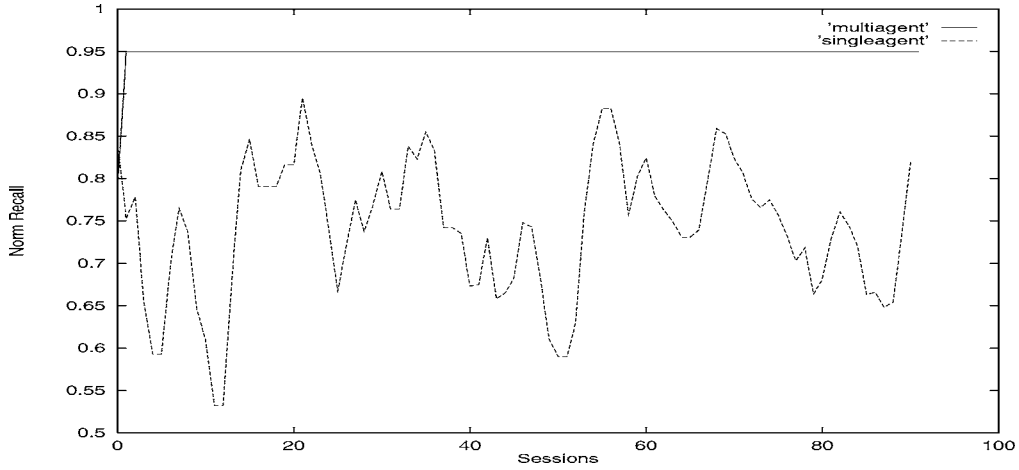

Figure 9. Comparison of the performance of Single and Multi agent.

them, D-SIFTER performed much better than the single agent scenario.

6.3. Scalability of D-SIFTER

In this experiment, we classified 5000 documents with different number of agents, ranging from 3 to 18. This experiment was performed to study the scalability of D-SIFTER and RMI with the increase in the number of agents. The time required to classify 5000 documents was observed. In this experiment, one agent was classifying 5000 documents and others were providing the remote assistance. The server was deployed on a Sun Ultra 5 machine and all agents ran on another Sun Ultra 5 machine. The result of this experiment is shown in table 2. As seen from the table 2, as the number of agents increases the classification time also increases in a polynomial manner. Two prominent factors contribute to this increase: a) these experiments employ the combined-opinion model, and b) the inherent slowness of Java RMI. Since all agents were located on one machine, they competed for resources, which resulted in an increase in the overall classification time. Eventually, when number of agents reached 18, the machine ran out of memory.

Table 3 shows the results of another experiment, almost identical to the previous one, except that the agents are deployed on different machines. This test was also performed using Sun Ultra 5 machines. It is no surprise that in the most of the cases the classification time required are less than the ones in the previous experiment (using a single machine to deploy all agents (using a single machine to deploy all agents)).

It should be noted that, like the previous experiment, the classification time in this experiment increases as the number of agents increases. This may seem surprising, especially when the agents are deployed on different machines, i.e., the addition of more computing resources has an adverse effect on the classification time. However, a closer look indicates that this is as expected, since the D-SIFTER prototype used for all these experiments employs the combined-opinion model of collaboration. As stated earlier, in this model of collaboration, all agents attempt to classify a document in a sequential order. Hence, when these agents are deployed on different machines, classification attempts by all agents translate into those many remote calls one after another. In the worst scenario, for each document needing a remote classification, $(n - 1)$ remote calls will be needed, where there are n agents in the system. Each remote call is expensive and hence, it is obvious that for a fixed number of documents, the increase in the number of agents results in a larger classification time. Although, the classification time does increase with the addition of more agents, the real benefit is the better success in classifying incoming documents – which is an enhances the quality of the filtering.

Table 2
Classification time for agents located on one machine.

No. of agents	3	6	9	12	15	18
Time (sec)	1525	3023	7611	17090	110607	Out of memory

Table 3
Classification time for agents located on different machines.

No. of agents	3	6	9	12	15	18	21	24	27	30
Time (sec)	1268	2039	5618	6751	8860	10757	13231	15330	17243	19702

Table 4 indicates the results of yet another experiment, in which all the agents are made to classify incoming documents. We increased the number of agents in the range from 3 to 30, and the number of documents from 2000 to 5000. Here, all agents participated in the remote document classification in addition to performing their own local classification. All agents and the server were deployed on different Sun Ultra 5 machines. It is obvious that for the same number of documents and agents, the total time required increased substantially as compared to the experiment where only one agent is classifying and others are assisting that agent.

Although, D-SIFTER has not been tested with a very large number of agents, these experiments suggest that D-SIFTER is scalable. In a real-world scenario, although the number of agents will be fairly large, it is unlikely that at any point, all the agents will be dealing with a large number of documents. Instead, the documents will be arriving at a reasonable rate and will be classified and presented to the user continuously. We are also in the process of testing D-SIFTER with much larger set of documents from the TREC collection (around 300,000 documents) and observing various parameters such as the number of centroids, the filtering performance, the overall response time, etc. The preliminary results indicate that D-SIFTER is able to handle such a large set of documents well.

7. Future work

D-SIFTER is an on-going effort at IUPUI. Many different extensions and enhancements are currently underway. In contrast to the homogeneous nature of the agents in D-SIFTER, we are are currently experimenting with a new architecture in

Table 4
Classification time.

No. of agents	3	6	9	12	15	18	21	24	27	30
2000 Docs.	1003	2983	5971	6023	6341	6548	6451	6622	6735	7029
3000 Docs.	1559	4050	8107	13473	13677	14022	14693	15074	15591	16073
4000 Docs.	2135	5203	10414	14214	14447	14373	16078	16152	16741	17057
5000 Docs.	2535	6433	12877	14674	14863	15784	16523	16899	17092	17556

which agents are heterogeneous. Each agent provides a specific service and the filtering is achieved by a loose coalition of a set of heterogeneous agents. Another effort is intended at employing complex economical models. We are also adapting D-SIFTER to different application domains. At present, D-SIFTER has been employed in three domains: Computer Science, Health Care, and News. As the design of D-SIFTER is OO-based, the migration to another application domain requires minimal changes to thesaurus and input-box modules.

8. Conclusion

Distributed systems are omnipresent in today's world. They offer many advantages such as higher performance and the exploitation of geographically dispersed computing/data resources. Despite these desirable features, the design of distributed systems is far from simple. The presence of multiple machines, interconnecting networks and partial failures add the complexity of a successful design of a distributed system. One way of taming this complexity is to follow the distributed object paradigm and use a language-dependent computation model, such as Java RMI, during the design and development of distributed systems. In this article, we have presented the design, based on Java RMI, of a distributed and multi-agent information filtering system. This system achieves a higher filtering performance by creating a collaborative network of filtering agents. The simplicity and portability of Java RMI have resulted in the creation of a robust prototype, which not only reduces the information overload on the user, but also exploits the net-centric computing infrastructure.

Acknowledgements

This project is supported by the Digital Library Initiative Phase 2. This is a joint program by multiple agencies: NSF, DARPA, NASA, NLM, LoC, NEH, FBI, NARA, SI, and IMLS.

References

[1] T. Berners-Lee, R. Cailliau, J. Groff and B. Pollermann, *World-Wide Web: The Information Universe, Electronic Networking Research, Applications and Policy*, Vol. 2, No. 1 (Meckler Publications, Spring 1992).

[2] G. Coulouris, J. Dollimore and T. Kindberg, *Distributed Systems: Concepts and Design* (Addison-Wesley, 1994).

[3] B. Kahle, An information system for corporate users: wide area information servers, Technical report, Thinking Machine Inc. (1991).

[4] F. Kilander, IntFilter home page (1996). http://www.dsv.su.se/ fk/if_ Doc/IntFilter.html.

[5] D. Lewis, Representation and learning in information retrieval, Dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst (1992).

[6] M. McCahill, The Internet Gopher: A distributed server information system, ConneXions, The Interoperability Report, Interop, Inc. (1992).

[7] J. Mostafa, S. Mukhopadhyay, W. Lam and M. Palakal, A multi-level approach to intelligent information filtering: Model, system, and evaluation, ACM Transactions on Information Systems 15(4) (October 1997) 368–399.

[8] A. Moukas, Amalthea: Information discovery and filtering using a multi-agent evolving ecosystem, in: *Proceedings of the Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology* (1996).

[9] K.S. Narendra and M.A.L. Thathachar, *Learning Automata – An Introduction* (Prentice-Hall, Englewood, NJ, 1989).

[10] R. Orfali and D. Harkey, *Client/Server Programming with JAVA and CORBA*, 2nd Ed. (Wiley, 1998).

[11] S. Peng, M. Mukhopadhyay, R. Raje, M. Palakal and J. Mostafa, A comparison between single-agent and multi-agent classification of documents, in: *Proceedings of IEEE Heterogeneous Computing Workshop, HCW 2001* (2001).

[12] R. Raje, Z. Liu, S. Chinnasamy, M. Zhong, W. Mascarenhas and J. Williams, Distributed-object computing, in: *High-Performance Cluster Computing*, Vol. 2 (Prentice-Hall, 1999) pp. 249–273.

[13] R. Raje, S. Mukhopadhyay, M. Boyles and A. Papiez, An economic framework for a web-based collaborative information classifier, in: *Proceedings of the International Association of Science and Technology for Development, SE'97 Conference* (IASTED/ACTA Press, Anaheim, CA, 1997) pp. 362–366.

[14] R. Raje, S. Mukhopadhyay, M. Boyles, N. Patel and J. Mostafa, On design and implementing a collaborative system using Java-RMI, in: *Proceedings of the 5th International Conference on Advanced Computing* (Tata–McGraw-Hill, New Delhi, India, 1997) pp. 404–411.

[15] R. Raje, S. Mukhopadhyay, M. Boyles, A. Papiez, N. Patel, J. Mostafa and M. Palakal, A bidding mechanism for web-based agents involved in information classification, World Wide Web Journal, Special Issue on Distributed World Wide Web Processing: Applications and Techniques of Web Agents 1 (1998) 155–165.

[16] R. Raje, S. Mukhopadhyay, M. Qiao, M. Palakal and J. Mostafa, A distributed information filtering system, in: *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics, SCI'2000* (2000).

[17] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer* (Addison-Wesley, Reading, MA, 1989).

[18] G. Salton and M.J. McGill, *Introduction to Modern Information Retrieval* (McGraw-Hill, New York, 1983).

[19] Sun Microsystems, Java™ remote method invocation specification, Technical report (1996).

[20] J.T. Tou and R.C. Gonzalez, *Pattern Recognition Principles* (Addison-Wesley, Reading, MA, 1974).

[21] T. Yan and H. Garcia-Molina, Sift – A tool for wide-area information dissemination, in: *Proceedings of the 1995 USENIX Technical Conference* (1995).

Rajeev Raje is an Associate Professor in the Department of Computer and Information Science at Indiana University Purdue University Indianapolis. Rajeev holds degrees from the University of Bombay (BE) and Syracuse University (M.S. and Ph.D.). His research interests are in distributed-object computing, component-based systems and software engineering. Dr. Raje's current research has been supported by the US Office of Naval Research, National Science Foundation and Microsoft Corporation. Rajeev is a member of ACM and IEEE (Computer Society).
E-mail: rraje@cs.iupui.edu

Mingyong Qiao is a software engineer with Kinko's Inc. He got his M.S. in computer science from IUPUI in 2000. His thesis was on distributed information filtering systems.

Snehasis Mukhopadhyay is an Associate Professor in the Department of Computer and Information Science and an Associate Director (Bioinformatics) in the School of Informatics at Indiana University Purdue University Indianapolis. Dr. Mukhopadhyay is a holder of degrees from Jadavpur University, India and the Indian Institute of Science, India as well as a Master of Science and Doctorate in electrical engineering from Yale University. Dr. Mukhopadhyay's research interests include Intelligent Systems, Intelligent Control, Neural Networks, Multi-Agent Systems, Intelligent Information Filtering, and Bioinformatics. He is a National Science Foundation CAREER Award recipient in 1996.
E-mail: smukhopa@cs.iupui.edu

Mathew Palakal is a Professor and Chair of the Department of Computer and Information Science and Director of Informatics Research Institute at Indiana University Purdue University Indianapolis. He received his Ph.D. in computer science from Concordia University, Montreal in 1987. His primary research interests are in pattern analysis and machine intelligence. He is working on problems related to information management using information filtering and text understanding approaches and structural health monitoring and smart diagnostics based on intelligent computational methods.
E-mail: mpalakal@cs.iupui.edu

Shengquan Peng is a software engineer with Verizon Wireless. He got his M.S. computer science from IUPUI in 2001. His thesis was on multi-agent systems.

Javed Mostafa is currently the Victor H. Yngve Associate Professor of information science at Indiana University, Bloomington. He holds a joint position of associate professorship in the School of Informatics and the School of Library of information science at Bloomington. He also has formal faculty affiliations with the Cognitive Science Program at IU, Bloomington (core faculty) and the Computer and Information Science Department (adjunct), at IUPUI, Indianapolis. Dr. Mostafa's research interest lies in intelligent user interface design for information retrieval. Current research projects include the Digital Libraries Phase II initiative of NSF (Award #9817572) and the Information Technology Research Phase I initiative of NSF (Award #0081944). He is member of the ACM and its Special Interest Group on Information Retrieval (SIGIR).
E-mail: jm@indiana.edu