Cooperative Matchmaking of Requests Among Distributed Change Monitoring Service Agents

Santi Saeyor
and
Mitsuru Ishizuka
Dept. of Information and Communication Engineering,
Graduate School of Engineering, University of Tokyo

ABSTRACT: Web repositories are huge amount of heterogeneous information sources. Besides the target information itself, the changes in Web repository are significant and worth being notified to the user who perceived the out of date information as soon as possible. The changes made in Web repositories occur at unpredictable rates. Unfortunately, stock type information source has no means to inform its prospective users about the changes. While the stock type information source occupies a large percentage of sources on the Web, it is necessary to have a system that monitors changes in Web repositories, and provides comprehensive presentation to the prospective users. This paper presents a system that provides such services with the cooperative matchmaking of requests in resource management module.

1. Introduction

The rapid growth of the interconnected information sources brings about overwhelming information, in addition, the information is supposed to be changed dynamically without any prior notification. A large number of information sources in Web repositories are stock type. The users access this type of information in pull mode, mostly by Web browsers. These information sources have no mechanism to bring information of the changes to prospective users.

We present the evolution of the distributed change monitoring service that notifies the users about changes in Web repositories. The improvement of overall utilization factor is derived from the resource management within each service unit and the cooperation among service units. The Matchmaking Agent is the key of coalition. The coalition among service units brings about a broader scope for request matching. Moreover, the Matchmaking Agent has potential to balance the services among service units. At the same time, the mechanism of load transfer based on coalition among the service units strengthens the robustness of the service. The decision making process relies on a cooperative game analysis among distributed users' representatives based on expected payoff values.

2. Architecture

The architecture of a unit of change monitoring service is shown in Fig. 1. The service is provided openly on the Web. Each user accesses the service via the Internet using any browser with Java Virtual Machine. Requests can be made directly to the *Service Agent* which is the front end of the service. The functions of

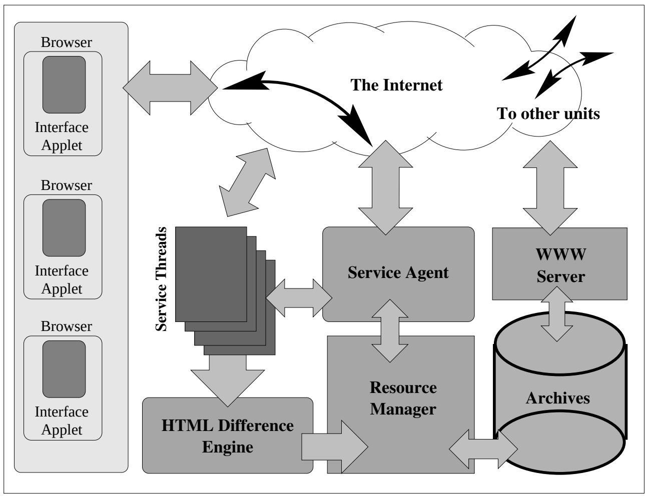

Fig. 1. *Each unit of monitoring service agent.*

main modules can be listed as follows:

- **Resource Manager**: All resources for the service are handled by this module. The results from the HTML Difference Engine will be kept in the archives by the Resource Manager. It tries to increase the effectively identical requests in order to improve the utilization factor, which is defined by the ratio of number of matched requests to the number of all requests.
- **Service Threads**: Each Web page monitoring request will be handled by a service thread. The thread keep monitoring and comparing revisions of the Web page.
- **Service Agent**: This is the heart of the service that interacts with users and other modules in order to retrieve and compare Web pages. The service unit keeps monitoring the changes in specified Web pages at specified intervals. When it is found that significant

changes are made on the pages, the Service Agent notifies the users about the changes and provides the comprehensive summary of changes in HTML documents.

- **HTML Difference Engine**: The service threads implement the Difference Engine in order to compare the content of updated pages and see whether there are significant changes in them. At the same time, it will summarize the updated information into another HTML document by an innovative algorithm described in [1].
- **WWW server**: The page archives contain the old and new version of Web pages together with summary pages. When the users are notified by the Service Agent, they can view the summary pages with their browsers via the WWW server.

The system is composed of a number of service agents distributed on the Internet as shown in Fig. 2. Each Service Agent announces the local service to the Matchmaking Agent in order that the identical or virtually identical requests can be relayed to the previously allocated services. The requests are served at

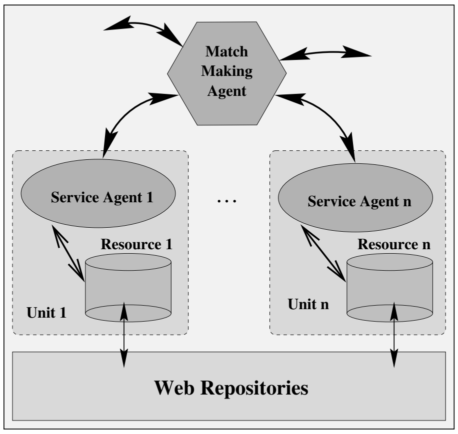

Fig. 2. *A number of service agents announce services to the Matchmaking Agent.*

arbitrary sites as shared services. Fig. 3. shows the life cycle of each service. The cycle begins when a user connects to a front end Service Agent. The requests made by the users are dispatched to appropriate Service Agents distributed in the network. The assigned pages will be monitored and compared to the old revisions by the service threads every period T. The changes will be assigned scores. If the scores exceed some thresholds, the changes are considered significant and will be notified to the users. The services terminate when service termination commands are issued by the users.

3. Resource management

When serving a large number of users, we expect

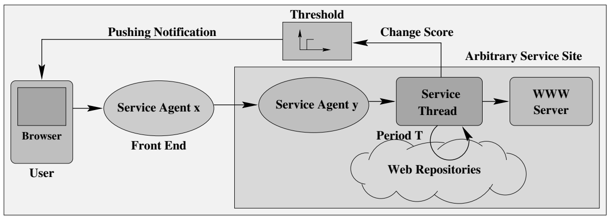

Fig. 3. *Life cycle of each change monitoring service.*

to have some identical or virtually identical requests. These requests can share the same resource. The more identical or virtually identical requests, the better utilization factor we can get from the service. The Resource Manager in each unit deals dynamically with the request matching. The conditions of virtually identical requests change dynamically upon the changes in Web repositories and the parameters of the requests made by the users. In the case of pushing changes

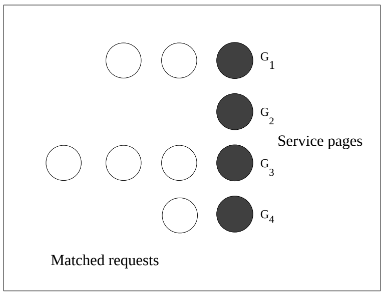

Fig. 4. *Groups of requests with the identical or virtually identical requests.*

and difference information, we push the information to prospective users. This implies that each user has different degree of interest and attitude against the detected changes. Fig. 4. shows that among different service pages, there are some identical requests. We can express the utilization of resource as following equations.

$$N = \sum_{i=1}^{M} G_i \quad (1)$$

$$M = P_{diff} N \quad (2)$$

$$\psi = \frac{N - M}{M} = \frac{1 - P_{diff}}{P_{diff}} \quad (3)$$

where:
N = number of all request
M = number of different kinds of requests
G_i = number of matched requests for the i^{th} group
P_{diff} = Probability of having different kinds of requests
ψ = Utilization factor

We can see obviously that if we share resource among users, we are likely to get more profit than serving each user separately. The utilization factor, finally, depends on the P_{diff} which ranges from $\frac{1}{N}$ to 1. The range tells us that our utilization factor ranges from 0 to $N-1$.

4. Matchmaking among service agents

Besides the self-organization in each service unit, the distributed service units work in coordination with the Matchmaking Agent as shown previously in Fig. 2. The coalition among Service Agents is made possible by the Matchmaking Agent. The agent has 3 main functions as follows:

1. **Request matching**: The agent tries to match nearly identical requests and dispatches them to appropriate Service Agent in the community.
2. **Issuing suggestions to users and Service Agents**: The agent has opportunity to evaluate the resource sharing efficiency because it holds the service profiles of the Service Agents in the community. It tries to adjust the load balancing and utilization factor of the overall system. The utilization factor can be improved if the agent can find new users who request nearly identical requests to the existing ones. The Matchmaking Agent also implements the same game analysis policy described in previous section to issue suggestions to the users.
3. **Providing information of services on demand**: Service Agents in the community may query the service profiles from the Matchmaking Agent. These profiles are necessary when a Service Agent deals with the users according to the access behavior of others.

The Matchmaking Agent cooperates with the Service Agents in the community by interpretation of incoming request objects as follows:

Request Object

The request object is a common template for requests posted to the Matchmaking Agent. A request object consists of requested function, input, output, owner, and time stamp. The functions currently implemented are listed as follows:

- **Declaration of service profiles**: In order to made the existing requests available to other service units, the Service Agent declares its service profiles to the Matchmaking Agent.
- **Load request**: When the Resource Manager considers that the load in the unit is still low compared to the available resources, it may give charity to the community by issuing a load request to the Matchmaking Agent. The constraint of the request object indicates the acceptable number of requests. The agent assigns requests to the service unit if available. The Matchmaking Agent returns the request object that the output was assigned a set of requests back to the service unit.
- **Load distribution request**: In some cases such as a service unit becomes overloaded or needs to be temporarily closed, the service unit has to distribute its services to others. It issues one or more request objects to the Matchmaking Agent in order to distribute the requests to available service units.
- **Effectively identical requests matching**: When a user request a service, the in charged Service Agent consults with the Resource Manager in the unit about an effectively identical request. In the case of no effectively identical request available, the Service Agent sends this request object to be served by the Matchmaking agent. In the worst case, there is no effectively identical request in the community, the Matchmaking Agent relays the request to an appropriate service unit. This can be both the service unit that issued a load request or the site which the Matchmaking Agent considers that the load is still low. However, the elected service unit has right to refuse the request base upon constraint on the location. If the request is not accepted by any service site, the Service Agent in charged takes the request into its own service unit.

Request Object:

Function:	*Request_function*
Input:	*Description, Constraint* *Input_obj*
Output:	*Description, Constraint* *Output_obj*
Owner:	*Service_Agent*
Time:	*Issued_time*

Fig. 5. *The template of a request object.*

Matching process

The Matchmaking Agent performs the request matching of incoming request objects. The agent investigates the function of each request and processes the input and output of the request according to the function. The request objects come to the agent in asynchronous fashion. In order to match the requests efficiently, the request objects should be interpreted at different priorities from the highest to the lowest: declaration of service profiles, load request, load distribution request, and request matching respectively.

The Declaration of service profiles should be rendered as fast as possible in order to provide the updated information about available requests. The load request should be rendered before load distribution request in order to register the demands. When the load distribution request or the supplies come to the

Matchmaking Agent, they can be distributed immediately. This comes from the fact that load distribution request has probability to be denied by any limited service units if load request is not available. The lowest priority is the request matching because this operation requires more processing effort and needs information of request with higher priorities.

When effectively identical requests are found, the users who request will be redirected to the service units where the requests are available.

Decision making in matching process

A cost model is applied to the service mechanism to investigate cooperative behavior of service agents in the community. The invited users under the invitation of preference adjustment requests and the new users gather together to get assigned to appropriate service groups. This implies that they desire to play in an assignment game in order to improve profits. Right in this process, the requests will be clustered into groups of prospectively identical requests. These groups are applied to a cost model for finding acceptable matches. The cost for a service is defined on the concept that the more effectively identical requests, the lower the cost of each service in the identical group. We define C_i, the cost per service for each user in group i^{th} based on F which is the full cost per user as follows:

$$C_i = F\left(1 - k\frac{e^{\frac{g_i}{N}} - 1}{e - 1}\right) \quad where \;\; 0 \leq k \leq 1 \qquad (4)$$

The cost reduction increases exponentially according to the number of identical requests in the group. This create a persuasion for the users to join a large group even the request for that group is not exactly matched what they want since this can reduce their costs. Fig. 6 shows the linear cost function and the cost function proposed above.

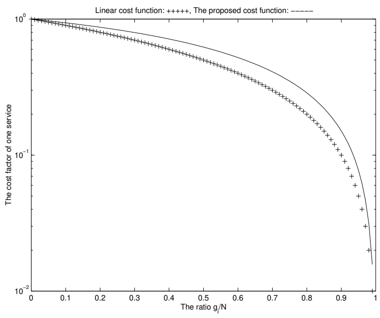

Fig. 6. *The cost function of a monitoring service.*

Another effect of the cost model is that the service costs in the community become lower remarkably as the number of identical requests increases. As a result, the competitiveness of the community becomes higher in case that we consider the service in multiple communities. The above is the evaluation of the value of services on the service provider side. The evaluation of services on the user side can be derived from the evaluation of distance between existing services and the service in need. We define the appreciation value of the user j^{th} against the service of the group i^{th} by u_{ij}. Then, the difference value for the service of group i^{th} viewed by the user j^{th} is $a_{ij} = u_{ij} - C_i$. If we have a set of service groups $G = \{g_1, g_2, ...g_n\}$, and a set of users $H = \{h_1, h_2, ..., h_m\}$, then we can find the maximum profit combination by finding the maximum value of $\sum a_{ij}$ among the combination space of G and H.

5. Summary

Decision making process of both Matchmaking Agent and Service Agent in each service unit relies on the game analysis. The expected payoff values based on experience in the past have direct impacts to the decision. The shared resource management based on game analysis and experience in the past helps us in providing information pushing service to a large group of user. The effort to improve the utilization factor of the service brings about an adaptive monitoring service. The cooperative behavior among users' representatives is formed according to the cost model. Thus, the system derives a better utilization factor. The cost model has a significant impact on the cooperative behavior and the overall profit. The effect of various cost models should be considered in future studies.

References

[1] Santi Saeyor and Mitsuru Ishizuka: WebBeholder: A Revolution in Tracking and Viewing Changes on The Web by Agent Community, in *proceedings of WebNet98, 3rd World Conference on WWW and Internet*, Orlando, Florida, USA, Nov. 1998.

[2] Fred Douglis, Thomas Ball, Yih-Farn Chen and Eleftherios Koutsofios: The AT&T Internet Difference Engine: Tracking and viewing changes on the web *World Wide Web* Volume 1 Issue 1, 1998. pp. 27-44

[3] Fred Douglis: Experiences with the AT&T Internet Difference Engine 22nd International Conference for the Resource Management & Performance Evaluation of Enterprise Computing System (CMG96), December, 1996.

[4] F. Douglis, T. Ball, Y. Chen, E. Koutsofios. Webguide: Querying and Navigating Changes in Web Repositories. In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996. pp. 1335-1344.

[5] Jeffrey M. Bradshaw: Software Agents AAAI Press/The MIT Press, 1997.

[6] Brian Starr, Mark S. Ackerman, Michael Pazzani: Do-I-Care: A Collaborative Web Agent *Proceeding of ACM CHI'96*, April, 1996.

[7] Aglet-Workbench - Programming Mobile Agents in Java, IBM Tokyo Research Lab.,URL=http://www.trl.ibm.co.jp/aglets/

[8] Kazuhiro Minami and Toshihiro Suzuki: JMT (Java-Based Moderator Templates) for Multi-Agent Planning *OOPSLA'97 Workshop*, 1997.