Formation of Cooperative Behavior among Information Agents in Web Repository Change Monitoring Service

Santi Saeyor Mitsuru Ishizuka

Dept. of Information and Communication Engineering, Faculty of Engineering,
University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, JAPAN
{*santi,ishizuka*}*@miv.t.u-tokyo.ac.jp*

Abstract. The Web repositories are exposed in form of a huge amount of heterogeneous information sources. The information is overwhelming the users. Besides the target information itself, the changes upon the previously released information are significant and worth being notified to those who perceived the out of date information as soon as possible.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 on the Web, and provides comprehensive presentation to the prospective users. In this paper, we proposes a mechanism that provides change monitoring and presentation service for a large group of users by coalition among service agents. The service agents keep improving the overall utilization factor by several schemes based on the decision made by game analysis. We apply a cost model to the service mechanism in order to study the cooperative behavior of the service agents. The reduction of cost is designed to comply with the level of cooperation among service agents. This paper presents a paradigm for the service and the formation of cooperative behavior of the agents in the community.

1 Introduction

Browsing through the sites for new updates is not only time consuming task but also vain in case that there is no change made on the sites once visited. This puts a significant load to the users besides exploring brand new information. We need some representatives to do such burdensome and tedious jobs for us.

This paper presents the evolution of mechanism that detects and evaluates changes on the Web, provides it in comprehensive form, and push the information to prospective users. The mechanism is evolved on multi-user basis. This paper proposes the coalition among service agents in order to maintain the performance both in the benefit of users and resource usage of the system. With this system, The ubiquitous stock type information sources on the Web have no need to provide any effort to convey their updates to the users.

We incorporate shared resource management in our system in order to enable the framework a larger scale of service. The service agents attempt to increase the utilization factor of the overall system by several schemes. Each scheme tries to increase identical services in the monitoring service. These identical or virtually identical requests give a significant impact on the utilization of our service. As long as we can implement the service with reasonable resource allocation, more users can have access to the service. A useful tool for decision making we use here is the game theory. We use the basics of game theory in several decision making situations during matchmaking process in resource management issue.

2 Distributed change monitoring

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

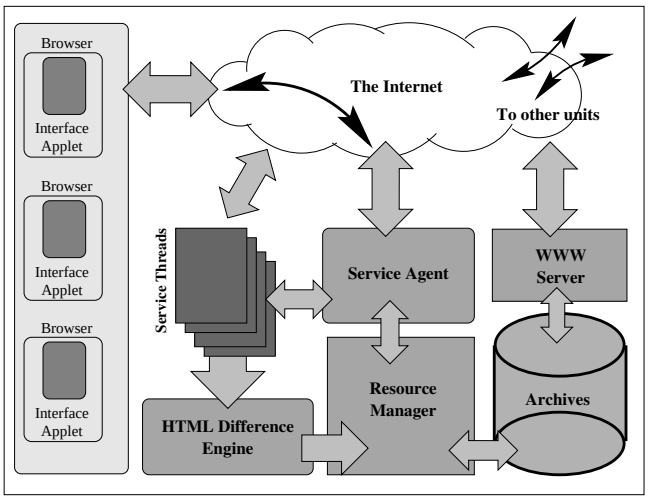

Fig. 1. *Each unit of monitoring service agent.*

of 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 Resource Manager.
- **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.
– **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 announce the local service to the Matchmaking Agent in order that if identical or virtually identical requests can be relayed to the allocated service. The requests are served at arbitrary sites

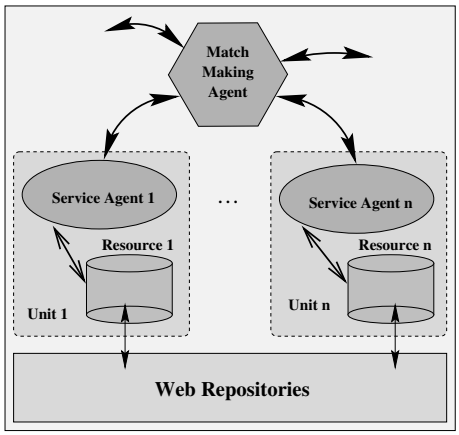

Fig. 2. *A number of service agents announce services to the Matchmaking Agent.*

as shared services. Fig. 3. shows the life cycle of each service. The cycle begins when a user connect to a front end Service Agent. The requests made by the users are dispatched to appropriate Service Agents distributed in the network. The assign pages will be monitored and compared to old revisions by the service thread 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.

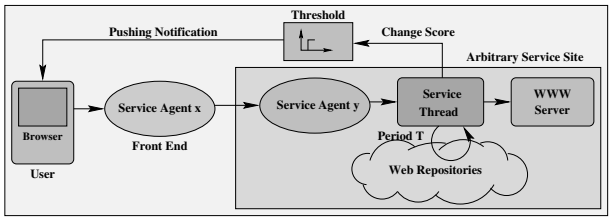

Fig. 3. *Life cycle of each change monitoring service.*

3 Resource management

When serving a large number of users, we expect 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 and difference infor-

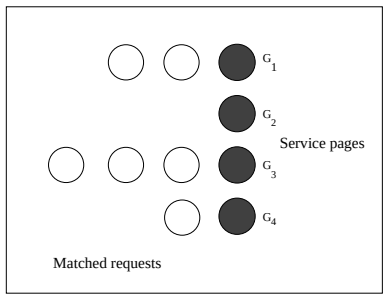

Fig. 4. *Groups of requests with the identical or virtually identical requests.*

mation, 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.

$$N = \sum_{i=1}^{M} G_i \tag{1}$$

$$M = P_{diff} N \tag{2}$$

$$\psi = \frac{N - M}{M} = \frac{1 - P_{diff}}{P_{diff}} \tag{3}$$

where:
N = number of all request
M = number of different kinds of requests
G_i = number of matched requests for 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 $N - 1$.

The amount of identical or virtually identical requests can vary dynamically. This is the case when some requests among currently identical requests are satisfied by the changed conditions but some are not. For examples, we decide to push changes information to the user if we found that the change score is higher than specific threshold points. Suppose we have 2 users who specified the score thresholds for an identical page at 1500 and 1000 points. Both requests are considered identical if the change score is 2000 points. However, if the change score falls between 1000 and 1500 points, the requests are no longer identical. The Resource Manager analyzes the characteristic of changes on Web pages. A

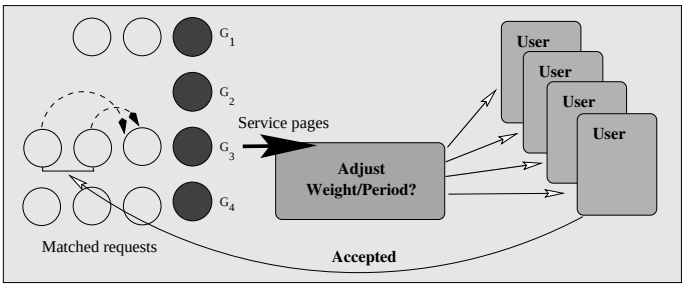

Fig. 5. *Increasing request hit rate by asking users to modify appropriate parameters.*

parameter to check is the notification threshold. In some cases, some users specified high thresholds with high frequency of monitoring. If the Resource Manager found that the change rates of those pages are relatively slow, it may ask the users to adjust threshold weight of notification or monitoring frequency. Adjusting these parameters has probability to increase more matched requests as shown in Fig. 5.

Meanwhile, the Resource Manager detects hot requests shared by a large number of users. The hot requests trend to be interesting pages. The Resource

Manager may recommend these requests to the users recorded in the Match-making Agent as shown in Fig. 6. If some users accept the recommendation, the

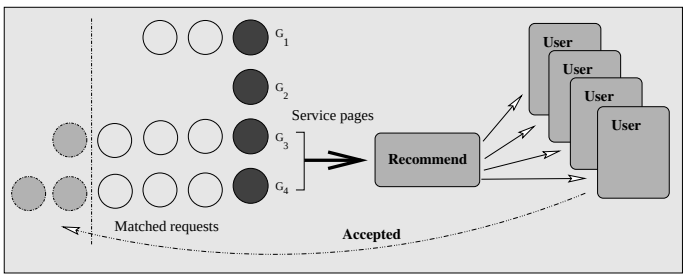

Fig. 6. *Increasing identical or virtually identical requests by recommending hot requests to the users.*

utility factor of the service trends to be increased according to the increasing matched requests. However, the Resource Manager has to make decision based on facts and experience in the past whether it should ask the users to adjust some parameters or recommend some hot requests to the users. The Resource Manager makes a decision by investigating the probability of improvement from the payoff matrix of the game. We consider how to make decision based on our service games from the following expected payoff matrix.

Table 1. Expected payoff matrix for the games of the service.

			User
		No Adj.	Adjust
	No Adj.	$< p_0, q_0 >$	$< p_1, q_1 >$
Service Agent	Adjust	$< p_2, q_2 >$	$< p_3, q_3 >$

The Payoff $< x_1, x_2 >$ indicates that the Service Agent has an expected payoff of x_1 (where an improvement is worth 1, and no improvement is worth 0 for the Service Agent) and the user has an expected payoff of x_2. In the case of our service $x_1 + x_2$ must be 1.

We define the utility as:
$$\begin{aligned} utility(ServiceAgent, 1) &\longleftarrow improvement \\ utility(ServiceAgent, 0) &\longleftarrow \neg improvement \\ utility(User, 1) &\longleftarrow improvement \\ utility(User, 0) &\longleftarrow \neg improvement \end{aligned}$$

Improvement occurs when:

$$improvement \longleftarrow ServiceAgent(D) \wedge User(D) \wedge improve_if_follow(D)$$
$$improvement \longleftarrow ServiceAgent(Adj.) \wedge User(NoAdj.) \wedge improve_if_sa_un$$

$$improvement \longleftarrow ServiceAgent(NoAdj.) \wedge User(Adj.) \wedge improve_if_sn_ua$$

Right here, the $improve_if_sa_un$ is the p_2 and the $improve_if_sn_ua$ is the p_1. Suppose that the Service Agent is to choose a strategy with $p_a = P_{ServiceAgent}(Adjust)$ and the user is to choose a strategy with $p_u = P_{User}(Adjust)$. In this setup, the probability of having improvement $P(improve)$ is defined by

$$\text{P(improve)} = p_a p_u p_3 + (1 - p_a)(1 - p_u)p_0 + (1 - p_a)p_u p_1 + p_a(1 - p_u)p_2 \quad (4)$$

In a randomized equilibrium from the Service Agent view, the payoff for $Adjust$ and $NoAdjustment$ must be equal. The payoff for asking for adjustment is the above formula with $p_a = 1$, the payoff for asking no adjustment is the formula with $p_a = 0$. These are equal, so we have

$$(1 - p_u)p_0 + p_u p_1 = p_u p_3 + (1 - p_u)p_2 \quad (5)$$

Similarly for the user

$$(1 - p_a)p_0 + p_a p_2 = p_a p_3 + (1 - p_a)p_1 \quad (6)$$

Solve for p_u and p_a we derive

$$p_u = \frac{p_2 - p_0}{p_1 + p_2 - p_0 - p_3} \quad (7)$$

and

$$p_a = \frac{p_1 - p_0}{p_1 + p_2 - p_0 - p_3} \quad (8)$$

Substitute in equation (4), we derive

$$P(improve) = \frac{p_1 p_2 - p_0 p_3}{p_1 + p_2 - p_0 - p_3} \quad (9)$$

The service agent checks whether the $P(improve)$ is above 0.5 which means the system has probability to improve the service more than 0.5, if it asks the user for adjustment. In our service, the probability p_3 can be derived by a function that evaluates how significant a request for adjustment is. The probability p_2 comes from the experience in the past which is, in other words, how much the user refuses the suggestion. The probability p_1 comes from the improvement made when the user adjusts the service profile without suggestion from the Service Agent. Finally, the probability p_0 comes from the self-improvement rate occurred as the conditions of changes in Web repositories vary in time domain. We can see obviously that the variables used in the game analysis above can be evaluated at ease from the statistic of the service. The Service Agent can make decision to deal or not to deal with the user by calculating the P(improve) based on transactions in the past. Moreover, each decision trends to be more exact as the experience of the Service Agent increases.

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. This profiles is 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:

4.1 Request Object

Request Object:

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

Fig. 7. *The template of a 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 become 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 in charged Service Agent takes the request into its own service unit.

4.2 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 highest to 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.

4.3 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 effective 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} base on F, the full cost per user, as follows:

$$C_i = F(1 - k\frac{e^{\frac{g_i}{N}} - 1}{e - 1}) \qquad where \;\; 0 \leq k \leq 1 \tag{10}$$

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 match what they want since this can reduce their costs. 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 value of the 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 Related work

From the standpoint of tracking and viewing changes on the Web, the works that are most similar to ours are that of Douglis, et al. [2],[3]. They used the AT&T Internet Difference Engine to compare revisions of Web pages of some enterprises from time to time. We attempt to improve the service for a larger scale of users. We manage the shared resources among users in order to enable induced push mode of changes and differences. Besides, our difference engine implements the *Longest Common Tag Sequence (LOCTAGS)* algorithm [1], which is capable of comparing context exactly where the revisions should be compared. The results from the HTML Difference Engine are comprehensive change presentation pages which precisely display down to the cell level of structured text.

In addition to a target page, we assume that the child pages are likely to have relevant information. Therefore, the service agent can be requested to watch the target down to its child pages. If needed, the agent can also be requested to watch deep down to the grandchild pages. However, the grandchild level is limited to the pages in the same domain of each child page.

The Do-I-Care agent [6] applies social discovery and filtering to inform the users when changes are detected. Moreover, it takes advantage of training the agent by groups of users where some interesting information may be offered to the user via the effort of others. We agree with the idea but we need a simpler way for evaluation of changes. The scoring method we use is straightforward and can be carried out quickly while providing a way for the users to adjust the threshold values upon their experiences [1]. In our system, the social filtering effect occurs when the Resource Manager cooperates with the Service Agent and the Matchmaking Agent in order to find hot requests to recommend to the users.

6 Summary

In this paper, we presented the mechanism of Web repositories change monitoring service that notifies the users about changes in Web repositories and creates comprehensive summary pages of the changes. 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 strengthen the robustness of the service.

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 users that are engaged to the service assignment game are grouped to appropriate service group based on cost model. The cost model promotes the degree of cooperation by compromising the user needs based upon the maximum profits. As a result, this increases the identical requests dramatically compared to matching only by similarity of incoming requests.

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. Imma Curiel: Cooperative Game Theory and Applications, Kluwer Academic Pubishers, 1997
8. Theo Driessen: Cooperative Games, Solution and Applications, Kluwer Academic Publishers, 1988
9. Aglet-Workbench - Programming Mobile Agents in Java, IBM Tokyo Research Lab.,URL=http://www.trl.ibm.co.jp/aglets/
10. Kazuhiro Minami and Toshihiro Suzuki: JMT (Java-Based Moderator Templates) for Multi-Agent Planning *OOPSLA'97 Workshop*, 1997.