International Journal of Multimedia and Ubiquitous Engineering
Vol. 2, No. 4, October, 2007

A Peer Common Interest Share Model

Dennis Muhlestein and SeungJin Lim

Department of Computer Science, Utah State University
{denjmuhle, lim}@cc.usu.edu

Abstract

On-line communities on the Internet are highly self-organizing, dynamic and ubiquitous. The prime interest of peers in this community is often sharing common interest, even when compromising privacy. This paper presents a peer coordination strategy and a data sharing process for peers on the Internet which allows them to discover their common interest in terms of sets of frequently visited URLs. To this end, an algorithm was developed that allows the user to collect URLs of common interest from peers who are currently visiting the same URLs as the user. Using the algorithm, sample data was collected by randomly following links on popular websites to simulate the algorithm in operation. Experiments were then performed to compare the number of discovered frequently visited URL sets and association rules with the overhead induced by our network. An extended discussion on the experiments performed is presented, by which the viability of the proposed model is demonstrated.

1. Introduction

On-line communities on the Internet are growing at an ever increasing rate. Not only is the number of people accessing these communities increasing, the methods for sharing data and information are changing [3]. Traditional sources of information are now being challenged by sources entirely made of user contributed content. New services that allow communities to determine which content is most relevant are becoming very popular. These services range from sites that allow users to vote on which news or information is most important to services that allow users to "tag" or label content. The relevancy of this user contributed data comes not from the ability given to each user, but from the combined input of many users. Recognizing the power of the community, Time Magazine named "You", the person of the year for 2006 [7].

It is safe to assume that the act of accessing a resource from a server, by a given user, denotes that the user has some level of interest in that resource [4]. We propose that the set of resources accessed by user A have some level of similarity with that of user B if A and B visited the same resource at some point in time. In other words, if two users have each accessed a set of resources that has some intersection, it is likely that each user will be interested in the portion of the other user's resources which they have not yet accessed. There is currently no quick mechanism on the Internet to gain access to that information.

The World Wide Web is made up of a large number of servers that are accessible on a global network. Clients connect to servers in a fairly unpredictable manner, e.g., it is a difficult problem for a web master to predict the IP address of the next client to access his/her server. Each server may maintain an access log that denotes the client, the time, and the resources that the client connected. This access log can provide insight into information such as the number of clients that accessed resources and the number of

resources accessed on the server, but the server has no knowledge of other servers accessed by any of its clients.

Web clients, on the other hand, keep a history of servers and resources that they visited. Web servers and web clients each have a portion of the entire log of accessed resources on the Internet. It is impractical and may not be possible to gather into one location the log data of all resources requested by all clients. Only this global access log in its entirety, if exists, can answer such questions as “what are the most commonly accessed resources on the Internet?" and “which resources are most commonly accessed in combination with other resources?" with a high degree of confidence and accuracy. Furthermore, because of the dynamic nature of the Internet in terms of the structure, the answers to these questions will constantly change.

Answers to the questions above could be approximated if the relevant data are processed in a distributed fashion where each peer in a distributed network has a small, but relevant, portion of the data.

This paper introduces a way for users to find information relevant to the data that they are currently interested in. The proposed method is unique in resource discovery: the user discovers potentially useful resources on the Web by peer collaboration based on the resource access behavior of peers, instead of by conducting usual keyword search through a search engine.

The rest of this paper is organized as follows: In Section 2, existing works related to our proposal are presented. We introduce an algorithm for peers to share and process data in Section 3. We then discuss our usage of sample data to test our algorithm and provide an analysis of our results in Section 4 followed by concluding remarks in Section 5.

2. Related work

The concept of knowledge discovery from web log data is not new. Much work has been done on different data mining algorithms for web usage data in the past. Several works deal with mining web logs to predict what the user will do next. In [8], the authors use the Apriori [1] algorithm to generate association rules which are then processed by a recommendation engine to suggest content to new users on a website. [5] also suggests pushing rules to a recommendation engine for processing but discovers rules based on clustering like users according the order of resources requested and the time spent on those resources. These methods generate rules with the intention of benefiting the web server or site operator. Our method, while taking a similar approach for mining data, aims to provide the interesting rules to the client for their benefit. In addition, our algorithm attempts to determine rules from all web usage across a community of peers as opposed to data for all clients from a particular web server.

[6] presents the idea of organizing web peers based on common interests. However, their work deals more with distributed data mining algorithms and only gives a broad overview of the idea. The authors propose having clients in a peer-to-peer network hold a majority rules vote to determine if a proposed frequent item set meets a minimum support/confidence. Our work has a similar intent but we organize peers based on commonly accessed resources instead of resource subject material. Relevant data is shared between peers, but then each client processes the information instead of determining rules on a network level. In addition, we propose a mechanism for coordinating peers.

3. The data share model

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of available servers on the Internet. Let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of all resources made available on the Internet. Let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of clients who are browsing available resources. Each client has a history of requested resources where the server and resource are identified, e.g., s_5r_3, s_2r_7, …. Servers on the other hand, store which resources were accessed by which clients, e.g., c_1r_3, c_2r_7.

Table 1 illustrates the difference in perspective between clients and servers for a hypothetical server s_1 and hypothetical client c_1. It is important to differentiate between the two perspectives. The server, while knowing which resources are requested from its domain by clients, cannot know about resources accessed on other server domains. The client on the other hand, while having access to resources on any domain, cannot know which resources are accessed by any other client.

Table 1. An example of server and client access log entries.

Log entries of server s_1	**Log entries of client c_1**
c_3r_2	s_1r_3
c_1r_3	s_2r_3
c_1r_4	s_1r_4
. . .	. . .

3.1. The coordination server

As pointed out earlier, the global access log is not available in one location for processing. It can however be simulated in a limited way by assigning portions of it to peer clients in a distributed network. The purpose of the proposed data share model is to help the web client find the access pattern of other clients across different servers. In our distributed network, this is achieved with the aid of a special type of web server, called a *coordination server* (CS).

Definition 1 (Coordination server) A coordination server is a web server which is designated to collect <client, server, resource, time> 4-tuples and serves a subset of the collection to its clients upon a request.

It is important that clients access the same coordination server when they access the same resource. The number of coordination servers for a particular on-line community is not important unless the community size is very large or the scalability of the system is critical. (For our tests we used one coordination server that is separate from any specific domain. This coordination server is responsible for coordinating all clients in our test network.) Practically however, it would be feasible to have the primary web server of a domain act as a coordination server. Peers are formed around the coordination server to which they request resources in our work.

Because the location of the coordination server isn't of consequence to the client's ability to operate in the network, there can be different mechanisms for clients to locate coordination servers. Perhaps a list of coordination servers would be made available much the same way lists of Usenet servers are published. The client might have the ability to select a server based on some criteria, like proximity, bandwidth, or simply user loyalty to the provider. Most likely, a plugin for the user's client program that implements the algorithm would be able to detect if the web server the client was accessing was acting as a coordination server (there could be an added HTTP response header for instance). If the web server was not participating as a coordination server, the distribution of the plugin would provide at least one coordination server for the plugin to use as a fallback.

When a client c_x makes a request to a resource s_ir_y on a particular web server s_i, it will then report to the corresponding coordination server, CS, the record $c_xs_ir_y$. The CS will then respond to c_x with the location of other most recent peers that also accessed s_ir_y. In other words, CS stores a temporary set of $\{c_xs_ir_y\}$ that

represents a portion of the most recent activity. Note that clients will use CS to locate peers, but do not necessarily have to report every requested resource $s_i r_y$ to CS.

The number of peers returned to the requesting client is configurable. The server makes no persistent connection with any of its clients so it has no way of knowing if the clients being returned are still available for data sharing. Internet clients by definition, can make requests to resources and then become immediately unavailable on the network. It may therefore, be best to return a small number of results if the time stamp of the accessed resources are within a short time frame. If however, a resource is rarely requested, it may be advantageous to the client to return a greater number of peers to ensure that the client can establish a connection with an optimal number of peers, i.e., we assume that some clients are no longer available if the resource is infrequently accessed.

The function of the coordination server is summarized in the following pseudo code:

[on client HTTP request *req*]
1. read resource r_y from *req* by client c_x;
2. store $c_x r_y$ and current time in local database;
3. select c from local database such that $c <> c_x$ and
   $r = r_y$ ordered by latest time stamp in descending
   order, limited to n results;
4. return $\{c\}$ to c_x;

Example 1 Consider a particular user c_x who is interested in politics. c_x has browsed various resources on the Internet and has come across a server with current information on a particular political view that he/she would like to learn more about. c_x has a plugin installed in their browser that communicates automatically with a coordination server CS. After c_x has requested this latest resource, say http://www.someserver/someresource, a request to CS is made. A sample HTTP request to CS and the response for c_x, i.e., the set of peer addresses that have requested the same resource lately, are shown below:

[request from c_x]
POST /visit HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: <length>
Connection: close

url=http://www.someserver/someresource

[response from CS]
HTTP/1.1 200 OK
Content-type: text/javascript
Content-Length: <length>

{‘192.168.1.1:8080', ‘192.168.1.2:8090'}

After the user's plugin receives a response from the CS, the plugin possibly has the addresses of one or more peers that have also requested the same server and resource. The plugin can now send and receive additional information from those peers as long as those peers are still available on the network.

3.2. Peer communication protocol

According to the server algorithm, clients are given a set of possible peers to communicate with as shown in Example 1. Because clients on the Internet can come and go at any time in an ad-hoc manner, no connections are persistent between these groups of peers. Each client has a lookup table with resources and peers, where each entry is a resource r_y corresponding to a set of peers $\{c\}$. For each resource the client requests, the client checks to see if it has at least one peer in the lookup table. If there are no peers for a particular resource, the client can request additional peers from the coordination server.

Example 2 Table 2 shows the hypothetical response by a coordination server for one common resource, namely r_x, between clients 4—7 assuming that the server considers 3 peers the optimal number of peers to respond with for this resource. Peer addresses are abbreviated using unique integers for convenience.

After a client receives a list of peers from the coordination server CS, or determines that it already has an adequate list available, the client sends the last requested resource, e.g., $c_x s_i r_y$, to all of its peers and those peers in turn add the client to their list of peers in the lookup table under the key r_x. (Our test client created a simple listen socket on a random port that was reported to the CS along with the IP address that the client was hosted at. To report a visited resource to a peer, the client made a simple HTTP post to its peer.)

Table 2. A hypothetical response by a coordination server to four clients 4, 5, 6 and 7 for a particular resource r_x.

Client	**Server response for r_x**
4	{1, 2, 3}
5	{2, 3, 4}
6	{3, 4, 5}
7	{4, 5, 6}

The function of a client is summarized in the following pseudo code:

```
1. create a set of locally visited resources;
2. create a lookup table of remote peers based on
resources;

[client visits a resource r_x]
3. append r_x to local visits in lookup table;
4. if lookup table does not have peer list for r_x then
5.   send a request containing r_x to CS;
6.   store any returned peer list in lookup table
       under key r_x;
7. endif
8. send r_x to all peers of r_x in lookup table;

[incoming resource r_y from client c_x]
9. append c_x to peer list in lookup table for key r_y;
```

Example 3 Consider the server responses in Table 2 of Example 2 again. Figure 1 illustrates the lookup table for the clients in Table 2. Client 4 receives peers 1, 2 and 3 from the CS with respect to the resource r_x. These peers are appended immediately to the lookup table of client 4 with resource key r_x. Client 4 then sends r_x to peers 1, 2 and 3. This process is repeated for clients 5, 6 and 7 (see Figures 2 through 4). After client 7 has contacted peers 4, 5, and 6, client 4 has peers 1—7 (excluding itself) in its lookup table for r_x as illustrated in Figure 4. If the process continued further, with additional clients requesting resource r_x, eventually each client would have 6 peers for the resource r_x.

3.3. Data share among peers by data mining

Peer	Resources requested by Peer
c_i	$\{r_{48}, r_{1447}, r_{2049}, r_{2050}, r_{2051}, \ldots\}$
c_j	$\{r_{57}, r_{58}, r_{62}, r_{64}, r_{136}, r_{137}, r_{764}, \ldots\}$
c_k	$\{r_{57}, r_{58}, r_{64}, r_{463}, r_{764}\}$
c_l	$\{r_{677}, r_{844}, r_{846}, r_{1544}, \ldots\}$
c_m	$\{r_{57}, r_{494}, r_{677}, r_{1634}, r_{1635}, \ldots\}$

Table 3. A sample lookup table for client c_x that has been rearranged into a table of transactions.

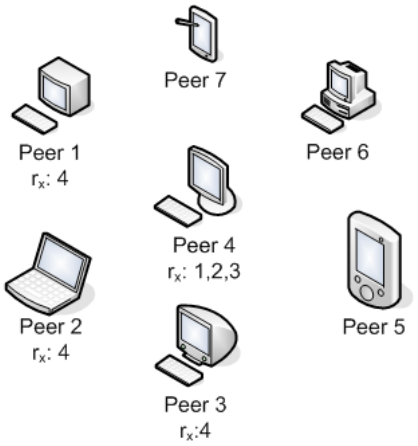

Figure **1.** Lookup tables after client 4 reported to CS, and contacted peers 1, 2 and 3 according to the response from CS. r_x: 1, 2, 3 at peer 4 implies that clients 1, 2 and 3 have requested the same resource r_x previously. Client 4 sends r_x to all of its peers (1, 2, and 3) and the lookup tables of clients 1, 2 and 3 are then updated as well

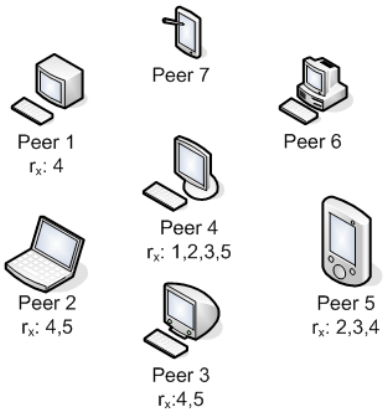

Figure 2. Lookup tables after client 5 reported to CS, and contacted p eers 2, 3 and 4 accordingly

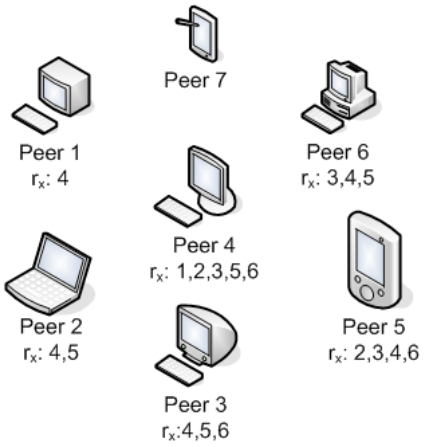

Figure 3. Lookup tables after client 6 reported to CS, and contacted peers 3, 4 and 5 accordingly

Based on the proposed server and peer communication protocol, it may be useful to find out which of the user's unvisited resources in the lookup table are most likely to be of interest to the user. To accomplish this, the client creates a list of resources visited by each unique peer in the lookup table. Each list of resources is seen as a transaction of items. In this step, the particular peer that requested the resources, nor the order in which the resources were requested is taken into consideration.

Example 4 After client c_x has contacted the CS several times, and other clients have become peers to c_x after requesting at least one of the same resources as c_x, c_x can begin to process the data. The client rearranges its lookup table into a list of peer transactions. Table 3 shows an example lookup table that has been rearranged into a transaction table.

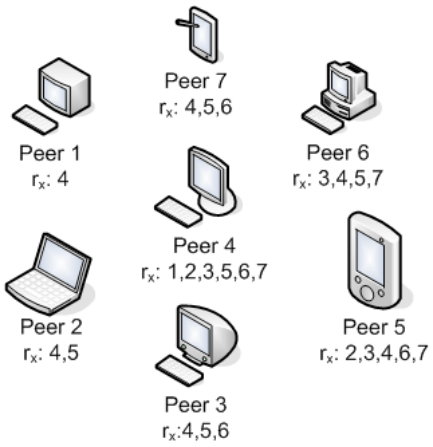

Figure 4. Lookup tables after client 7 reported to CS, and contacted peers 4, 5 and 6 accordingly

This list of transactions can be scanned for frequent item sets and association rules [1]. The frequent item sets provide insight into which resources are commonly accessed together by peers. The association rules help the client decide which previously unvisited resources are most interesting when taking the access patterns of its peers into consideration.

The most interesting association rules [9] for each client are rules that contain the most items from the client's local visits in the rule's antecedent. The consequent of these rules suggest to the client resources that they may be interested in, i.e., peers who visited the same resources this client visited, also visited these resources that this client hasn't yet visited. In other words, all the rules that are in the form $\{r_1, r_2, \ldots, r_n\} \rightarrow r_z$, where r_i ($1 <= i <= n$) were resources visited by the client, suggest with some degree of confidence that r_z is also of interest to the client. Clients are not interested in rules that contain locally visited resources in the consequent (the client already requested that resource). This is a good heuristic for limiting association rules and can be applied after generating frequent item sets and before searching for association rules. The search space for association rules is limited by disallowing locally visited resources in the consequent and disallowing all other items in the antecedent. After applying these two conditions, all rules found are in the format $\{ r_i \} \rightarrow r_x$, where r_i is one or more locally visited resources and r_x is a resource visited by a group of remote peers who also visited $\{ r_i \}$. After evaluating all discovered rules, the most interesting rules are the rules that

1. have the most items in the antecedent,
2. have the highest support, and
3. have the greatest confidence.

We prefer rules that have the most items in the antecedent (all of which the client has requested locally) since they may indicate that the consequent resource is more strongly associated with the antecedent resources. Rules that have a larger support indicate that more peers visited the same group of resources. A greater confidence indicates that more of the peers that visited the antecedent resources also visited the consequent resource. Each of these conditions can be evaluated individually or in combination when determining which resources to suggest to the client.

Example 5 Consider Table 3 of Example 4. After scanning the table for frequent item sets, c_x determines that the set $\{r_{57}, r_{58}, r_{64}\}$ meets the minimum support requirement and consists of resources

this client has visited. It next determines that r_{764} meets the minimum confidence for the association rule $\{r_{57}, r_{58}, r_{64}\} \rightarrow r_{764}$. The rule is kept for later consideration because c_x has not visited r_{764}.

It is likely that the confidence of a discovered rule is less than 100%. In other words, not all clients who visited the antecedent resources of the rule, also visited the consequent resources. A client can forward a discovered rule to all peers who have visited the antecedent resources but not the consequent resource. This may be of value to the client's peers because peers do not necessarily find the same rules. Note that peers are not transitive. Just because clients c_1 and c_2 are peers, and c_2 and c_3 are peers, c_1 and c_3 are not necessarily peers.

4. Experiments

The global access log is not currently available. There are some services currently in operation that use a browser plugin to transmit visit data from the client to a 3rd party service but the raw data is not made available for download. There are some data sets for web traffic analysis but they are limited to one particular domain.

We implemented our coordination server as a stand alone web server written in Python. Our CS uses a PostgreSQL database for temporary storage. To test the client, we developed a plugin that is installable in the Mozilla FireFox web browser. To test the applicability of our algorithm, we needed to generate sample data. Our sample data was generated and tested with software written in Python.

We wanted to simulate a certain number of clients visiting resources on the Internet and record each client's local log file for analysis. We selected the top 100 websites based on the ranking provided by Alexa.com [2]. The test client would choose a random website from this list and retrieve the content located at the root of this web server. The client would then record the requested resource and then choose a new resource randomly from the links within this web page. This was performed in a loop with a random chance of continuing.

The following pseudo code illustrates how sample data was generated in our experiment:

```
select random site from top 100 sites;
set current domain to randomly selected site;
set current resource to /;
loop
  record current domain/resource in log file;
  select random number r in [0,1];
  if r > chance of exiting then exit;
  retrieve and cache current domain/
    resource;
  randomly select new domain and new
    resource from all links on retrieved
    content;
```

This client outputs one record of visits per row in the log file. After running this client many times, the log file contains records that represent many users browsing at the same time. The next step is to convert the log file into data that represents each client access in real time. Assuming that all clients start browsing at approximately the same time, the data can be converted into a new log file with the following algorithm.

Conversion algorithm:

```
create ordered list of clients/visits keyed from
     current visit time;
for each line in sample data log:
  current time=0;
  for each visit:
    select random integer r in (3,10);
    current time += r;
    visits[current_time] = current client/visit;
output ordered list;
```

Sample output:

$c_7 s_3 r_9$
$c_{29} s_4 r_1$
$c_3 s_5 r_2$

The output of this algorithm converts the former log file into a log file with 1 client/visit per line. Clients and visits can appear in any order. This is somewhat indicative of what happens on the Internet. Time is unimportant for the simulation though because the simulation results will be the same no matter how long between visits each client takes. The important factor is the order that clients make their visits. This is because the coordination server selects peers based on whom has most recently accessed a server resource. It is important to note that in a real network environment, the time does start to play a factor because old peers may become unavailable.

4.1. Analysis of sample data

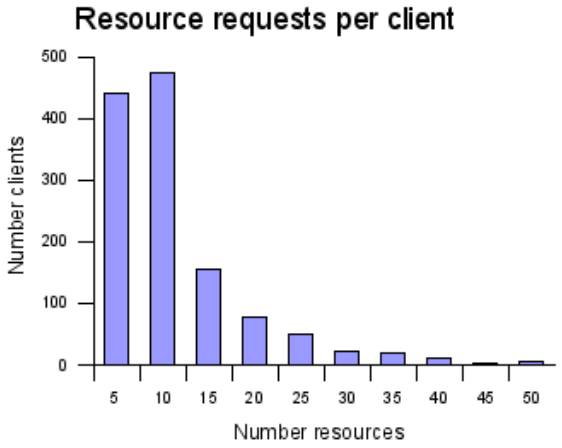

Figure 5. Number of resource requests per client after rule sharing in our test network.

The sample data contains resources requested by 1,279 clients. These clients started by randomly choosing one of 100 sites. From there however, no restrictions were placed on which resources and servers were contacted. A total of 1,086 unique server domains were contacted and a total of 6,320 unique resources were visited. There were 11,578 total requests to resources (which yields 9 requests per client in average). Figure 5 shows a number of resources, and how many clients contacted that number of resources. The average is 9 resources/client and the mode is 6.

Because we used only one coordination server and started from a relatively small number of domains, all of these clients are part of the same group of peers, meaning that one could follow references from one peer to another, and eventually traverse to all peers in the network. In Figure 6, we show for each number of peers, how many clients in our test network contacted that number peers. The average is 10.7 peers/client and the mode is 12. In our test network, each client contacted 10.7 peers in average.

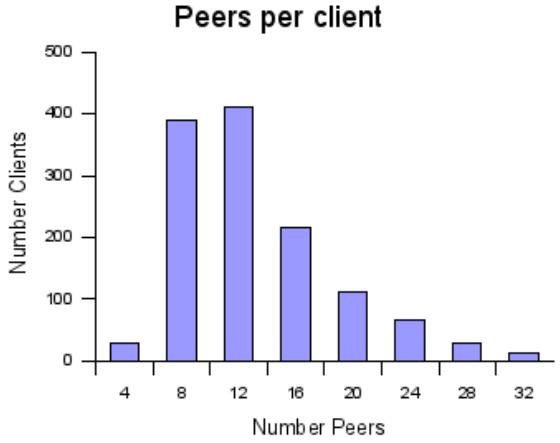

Figure 6. Number of peers per client after rule sharing in our test network.

Our test program used the larger of 3 divided by the total number of transactions known by the client or 10% for the minimum support percentage and a minimum confidence of 50%. With these values, 57% of all clients found interesting rules with an average of 2.18 rules per client found. Of these rules, on average, 2.11 rules were found to be interesting for at least one peer. After distributing interesting rules amongst peers, 86% of all clients had access to at least one rule with an average of 5.51 rules/client. Figure 7 shows the number of rules and the number of clients that either discovered or were presented with that number of rules.

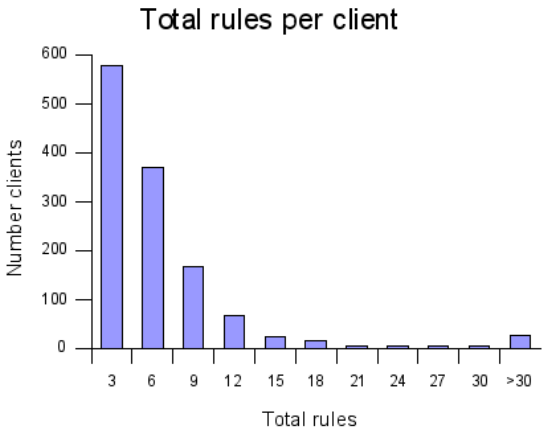

Figure 7. Number of rules per client after rule sharing in our test network.

In our test network there were 125 clients that found exactly 3 interesting rules. To evaluate the effectiveness of our data sharing protocol, we measured the average number of peers, locally visited resources, and number of resources received from these 125 peers each time a new association rule was discovered. Table 4 contains the cumulative averages for these clients. The same information is displayed in Figure 8 for clarity.

	Vistis	Peers	Received Resources
1st rule	5.0	7.8	28.0
2nd rule	7.0	10.3	47.9
3rd rule	8.4	11.2	65.6

Table 4. The average number of visits made, peers identified and received resources from the peers by the time each rule was discovered by the clients who discovered three rules.

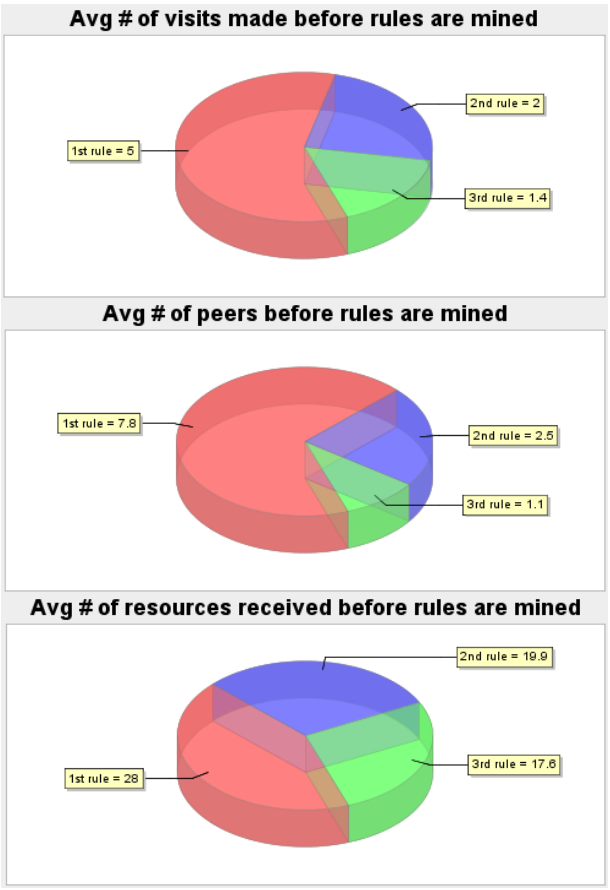

Figure 8. The average number of visits made, peers identified and received resources from peers shown in Table 4

From the averages listed in Table 4, we can see that there is an initialization cost before a client receives a benefit from the network. On average, a client didn't find a rule until they had visited approximately 5 resources and made a connection with 7 or 8 peers. After this cost however, discovery of new rules took less network communications as it only took 3 to 4 more local visits to resources and 4 more peers to discover an additional two rules on average.

5. Conclusion and future work

We can see, that for the group of peers in our test network, by the time a client has requested 10 resources, it can be presented with another 5 or 6 resources which may be of interest to the particular client. Since this is an ongoing process as clients browse, rules may be found at any point in time. We believe that given enough clients in a network using our system, each client would be able to continually find new resources of interest, which can be considered as an alternative Web resource discovery method to popular keyword search using a Web search engine.

Our peer-to-peer model relies on a coordination server. Even though this component of the network can be fulfilled by many actual servers, there would be benefits to creating a peer lookup model that didn't rely on centralized servers.

In our current model, the weighting scheme for peer resource access behavior is naive. In other words, every peer resource access is weighted equally. In order to refine the quality of common interest, it would be worthwhile to distinguish serious, intentional resource access from casual, random access.

Our model for sharing data mining focuses on connecting peers that have similar interests. The data mining portion simply runs the Apriori algorithm on the collected information. It may be advantageous to use a distributed association rule mining algorithm instead. Perhaps instead of sharing resources, clients could share association rule candidates with their peers to determine rules of interest.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami, “Mining Association Rules between Sets of Items in Large Databases,” In Proceedings of the 1993 ACM SIGMOD Int'l Conf. on Management of Data, pp. 207-216, May 1993.
[2] Alexa.com, “Alexa Web Search - Top 500,” http://www.alexa. com/site/ds/top_sites? cc=US&ts_mode=country, Last retrieved December 2006.
[3] R. Blood, “How blogging software reshapes the online community,” Commun. ACM, Vol. 47, No. 12, pp. 53-55, 2004.
[4] M. Claypool, D. Brown, P. Le, and M. Waseda, “Inferring User Interest,” IEEE Internet Computing, Vol. 5, No. 6, pp. 32-39, 2001.
[5] S. Gunduz and M. T. Ozsu, “A Web page prediction model based on click-stream tree representation of user behavior,” In Proceedings of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data mining, pp. 535-540, 2003.
[6] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta, “Distributed Data Mining in Peer-to-Peer Networks,” IEEE Internet Computing, Vol. 10, No. 4, pp. 18-26, 2006.
[7] L. Grossman, ‘Time's Person of the Year: You,” Time Magazine, December 2006.
[8] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, “Effective personalization based on association rule discovery from web usage data,” In Proceedings of the 3rd Int'l Workshop on Web information and Data Management, pp. 9-15, 2001.
[9] R. J. Bayardo, Jr. and R. Agrawal, “Mining the most interesting rules,” In Proceedings of the 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data mining, pp. 145-154, 1999.

Authors

Dennis Muhlestein

Dennis received a B.S. in Computer Science from Utah Valley State College, USA in 2002. He is currently pursuing an MS degree at Utah State University and works as a software engineer and network administrator. His interests include Distributed Computing, Algorithms, Artificial Intelligence, Data Mining and Peer to Peer technology.

SeungJin Lim

Received a B.S. degree in Computer Science from University of Utah, USA, 1993, and M.S. and Ph D degrees in Computer Science from Brigham Young University, USA, in 1995 and 2001 respectively. In 2003 he joined the faculty of Utah State University, USA where he is currently an assistant professor in Department of Computer Science. His research interests include mainly Data Mining.