A Modified Competitive Agglomeration for Relational Data Algorithm

Lisa Gandy, Shahram Rahimi, Bidyut Gupta
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL 62901, USA
[lgandy, Rahimi, bidyut]@cs.siu.edu

***Abstract* - Clustering algorithms are invaluable methods for organizing data into useful information. The CARD Algorithm [1] is one such algorithm that is designed to organize user sessions into profiles, where each profile would highlight a particular type of user. The CARD algorithm is a viable candidate for web clustering. However it does have limitations such as long execution time. In addition, the data preparation for the algorithm's requirements employs concepts that are incomplete. These limitations of the algorithm will be explored and modified to yield a more practical and efficient algorithm.**

I. INTRODUCTION

As the World Wide Web has grown in size, an immense amount of data has been produced. If this data is not retrieved and organized correctly then the information that the data can provide is essentially wasted. Users often become frustrated because they are certain that the information that they are looking for is on a particular website, but the website is so large that it is almost impossible to find the information. In this case, an adaptive website can become very helpful. This website would follow the *clickstream* of the user. That is, the website will keep account of web pages that the user is looking at, and recommend other pages that previous users found useful.

The website will implement this adaptability by having a database of typical user profiles. A user profile is defined as an "abstract model that summarizes the relevance of each URL on a site relative to a group of users sharing a similar interest [2]." The user profile will be characterized by previous user sessions. A user session is a set of web pages that a user examined within a specified time period. To organize the user sessions into profiles, the website administrator could examine each user session and manually place them into profiles. However, this is unrealistic for many reasons, for instance, many web sites have millions of users accessing it daily and it is impossible to find patterns by mere manual examination. Therefore, we can easily see that an unsupervised clustering algorithm is an attractive choice for clustering user sessions into profiles of several types of typical users since it relies on user access patterns and is capable of examining large amounts of data in a fairly reasonable amount of time [1].

Although the CARD algorithm has found useful user profiles in prior publications, there are several assumptions made during the implementation of the algorithm that are questionable [1]. For example, when the relational data is being formed, URL paths of HTML documents are compared. When forming similarity data the authors assume that similar web pages will be in similar folders. Ideally this might be the case, but generally the World Wide Web can be quite disorganized and pages that are in the same folder might not have the same purpose. We propose comparing web pages with an algorithm that will scan the title of the page, and then scan for keywords. The keywords of the two documents will then be compared and a similarity in the range of [0, 1] will be found. A comparison will then be implemented to find whether the new clusters that result from the modified data are more valid.

In addition, the original algorithm takes a large amount of time to run. From our initial experiments with 1800 web sessions, the algorithm took approximately 6 hours to run. This of course could be done offline, but if actually used, it would take an entire night to formulate user profiles. If the number of web sessions grow, then the time to execute will grow exponentially and quickly become very unreasonable. Therefore we will analyze methods to decrease the execution time of the algorithm when run serially, and also implement the algorithm in a parallel computing environment. The algorithm easily lends itself to parallel computation since it is primarily based on matrix mathematics.

II. COMPETITIVE AGGLOMERATION FOR RELATIONAL DATA ALGORITHM

The CARD algorithm is characterized by (1):

$$J(\mathbf{U},\mathbf{B};X) = \sum_{i=1}^{C}\sum_{j=1}^{N}(u_{ij})^2 d^2(x_{j,}\beta_i) - \alpha\sum_{i=1}^{C}\left[\sum_{j=1}^{N}u_{ij}\right]^2 \qquad (1)$$

subject to $\sum_{i=1}^{C}u_{ij} = 1$, for j $\in$ $\{1,,\cdots,N\}$.

where $X = \{x_j \mid j = 1,\ldots N\}$ is a set of N vectors and B = ($_1$,... $_c$) corresponds to C-tuple prototypes, where each prototype characterizes a cluster. In (1), $d^2(x_j,\ _i)$ represents the distance from the feature vector x_j to the cluster $_i$ and u_{ij} represents the membership of the point x_j to the cluster $_i$. The first term given in the equation controls the size and the shape of the clusters. The second term controls the number of clusters [1].

The clustering algorithm can be used for feature vectors - in this case it is simply known as the CA (Competitive Agglomeration) algorithm. However for several reasons, as discussed in [1], clustering user sessions requires the use of relational data, which can be represented by a matrix of similarities between each session to all other sessions.

To define the similarity matrix of size $N_s x\ N_s$ (where N_s is the number of user sessions) we first assume that each session is defined by a vector known as s which has size N_{url} (where N_{url} is the number of URLs) . Each entry contained in the vector can either contain a 1 or 0 depending on whether i^{th} URL was visited during the particular user session. Thus, the relation between two user sessions can be found by applying (2), which is essentially then number of identical URLs between the two sessions related to the number of URLs used in both sessions [1].

$$S_{1,kl} = \frac{\sum_{u-1}^{N_{url}} s_i^{(k)} s_i^{(l)}}{\sqrt{\sum_{i=1}^{N_{url}} s_i^{(k)}} \sqrt{\sum_{i=1}^{N_{url}} s_i^{(l)}}} \quad \textit{for } k=1..N_{s,}\ l=k..\ N_s \tag{2}$$

Equation (2) relies on the fact that only identical URLs between sessions are considered. However, two web pages might not have the same URL but may have quite similar content or usage. Past research has suggested that the syntactic similarity of two URLs should be considered. This syntactic similarity will be known as S_u, and is displayed in (3) [1].

$$S_u(i,j) = \min\left(1, \frac{|p_i \cap p_j|}{\max\left(1, \max\left(|p_i|, |p_j|\right) - 1\right)}\right) \tag{3}$$

This similarity will lie between [0,1]. For instance the web page www.cs.siu.edu/~cs491-3/index.html and the web page www.cs.siu.edu/materials.html would have a syntactic similarity of 0.5. This syntactic similarity is then incorporated into the intersession similarity equation to form the updated (4) [1].

$$S_{2,kl} = \frac{\sum_{i=1}^{N_{url}} \sum_{j=1}^{N_{url}} s_i^{(k)} s_j^{(l)} S_u(i,j)}{\sum_{i=1}^{N_{url}} s_i^{(k)} \sum_{j=1}^{N_{url}} s_j^{(l)}} \tag{4}$$

However, it can be shown that when the syntactic similarities are low S_1 is a better approximation of session similarity and then the syntactic similarities are high S_2 is a better approximation. Therefore to get the optimal similarity measure past research suggests that the maximal result of the two equations should be taken as S_{kl}, as shown in (5) [1].

$$S_{kl} = \max(S_{1,kl}, S_{2,kl}) \tag{5}$$

For further information concerning the CARD algorithm please refer to [1].

III. Improving Clustering: The First Modification

A. Theory

As stated previously, finding S_u by comparing URL paths can be problematic, in the sense that simply because two web pages are in the same directory structure does not guarantee that they will have similar content. I proposed comparing each document to all other documents and finding a keyword similarity that will be added to the similarity found in (3). The proposed keyword similarity is given in (6)

$$S_{key}(i,j) = \frac{\sum_{k=1}^{Uniqkey} \min(key_k^{(i)}, key_k^{(j)})}{\sqrt{\sum_{k=1}^{Nkey(i)} key_k^{(i)}} \sqrt{\sum_{k=1}^{Nkey(j)} key_k^{(j)}}} \tag{6}$$

UniqKey identifies the number of matching keywords between two documents *i* and *j*. The identifier $key_k^{(i)}$ represents the number of occurrences of keyword *k* in document *i*. Therefore in the numerator of (6), a running total of the minimum occurrences of matching keywords in the compared documents is taken. The denominator of (6) is simply the number of occurrences of all keywords occurring in documents *i* and *j*. This equation has the attractive property that $S_{key}(i,i) = 1$ and $S_{key}(i,j) = S_{key}(j,i)$ and $S_{key}(i,j) = [0..1]$.

The use of *min* in the numerator deserves some thoughtful consideration. The idea is that if a keyword occurs very seldom in one document and very frequently in another document then the two documents must not be very similar in general. In this case it is better to take the smaller value of keyword frequency to reflect this dissimilarity. If the average was taken then the fact that one document might have a small frequency for one keyword and the compared document has a large frequency might be masked.

Once the keyword similarity is found it is added to the similarity found in (5) to find a new similarity measure given below in (7). When the combined values of *S* and S_{key} exceed 1, we simply choose 1 as the maximum value. Notice that S_{key} has been given a weight of two thirds and *S* has been given a weight of one thirds. In previous trials S_{key} was simply added to *S*, however at these times, the clusters formed were too large, because all pages had very high similarity measures. At the same instance, however, we want the keyword similarity to have more weight than the URL similarity so that when comparing using only URL weights and URLs and keywords weights combined, we see a noticeable difference in cluster size and shape.

$$S_{ukey} = \min((S + S_{key} * 2)/3, 1) \tag{7}$$

B. Implementation

During keyword parsing the Porter Stemming Algorithm is used. The purpose of this algorithm is to remove suffixes that can cause keywords to seem different when they actually have the same meaning. By using this algorithm the performance of the keyword system will improve, and will also be less complex, because there will be less keywords per document [7]. The algorithm used is taken from the URL http://www.tartarus.org/~martin/PorterStemmer/Java.txt. After all keywords are found, keyword similarity is generated and added to the URL similarity mentioned previously.

We should keep in mind that the keyword algorithm that was used is not the most state of the art keyword processing algorithm that can be found. It simply considers words between certain html tags (such as <p>, <a>, <title>, <b> and <i>), gleans out words such as possessives, articles and commonly found scripting terms and then stems these words. No document term frequency is found. More advanced keyword processing was not possible due to time constraints. However with further research enhanced keyword processing should give enhanced results.

C. Results

1) Performance: One of the key points of concern when adding keyword similarity to the generation of similarity data using URL syntactic similarity is that this process will add too much execution time to the overall algorithm. Although this computation did incur additional execution time, this time was negligible. For instance before intersession similarity can be found using keywords, the actual keywords must be parsed from the html documents that have been referenced by the users. With 1116 user sessions and 2151 html documents it took approximately 40 seconds to find all keywords. Fig. 1 illustrates the amounts of time that it took to find keywords for varying numbers of user sessions. After all keywords have been found, then the keywords can be used to find intersession similarity. The time between finding intersession similarity when using just URLs and when using URLs and keywords to generate intersession similarity is also low. For instance with 1116 user sessions and 2151 URLs it took approximately 58 seconds to find similarity using URLs and approximately 66 seconds to find similarity using URLs and keywords. Fig. 2 illustrates the time differences between using URLs and URLs and keywords for different numbers of user sessions.

2) Cluster Validity: Before results of using URL similarity and combined URL and keyword similarity can be examined, the proper validity measures must be chosen. For this paper, the equation that finds clusters is given in (8).

$$X_i = \{s^{(k)} \in S \mid d_{ik} < d_{jk} \quad \forall j \neq i\} \qquad for \quad 1 \le i \le C \tag{8}$$

Once we find the clusters using (8), then we find the intra-cluster similarity. This is a cluster validity measure that gives the approximate distance between all pairs of sessions in a cluster. The formula used to find the intra-cluster distance is given in (9) [1].

$$\bar{D}_{Wi} = \sum_{s^{(K)} \in X_i} \sum_{s^{(l)} \in X_i, l \neq k} d_{kl}^2 / |X_i|(|X_i|-1) \tag{9}$$

Another useful validity measure for clusters is that of URL probability per profile, given in (10) [1].

$$P_{ij} = p(s_j^{(k)} = 1 \mid s_j^{(k)} \in X_i) = \frac{|Xi_j|}{|Xi|}, \text{ where } X_{ij} = \{s_j^{(k)} \in X_i | s_j^{(k)} > 0)\} \tag{10}$$

P_{ij} represents the probability that URL j will occur in cluster i, where cluster i is represented by X_i. For cluster validity, we tend to favor clusters where the intra-cluster distance is low and the majority of URL probabilities per cluster are high. If the URL probabilities are high then there is a frequent pattern of URL access. A "bad" cluster will generally have high intra-cluster distance and low URL probabilities.

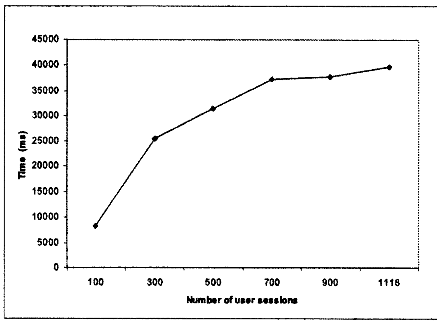

Fig. 1 Time to find keywords for varying number of user sessions

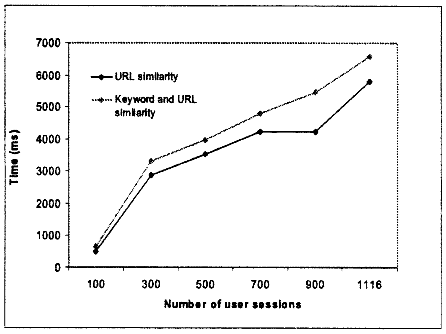

Fig. 2 Time to find intersession similarity using just URLs and URLs and keywords

3) Discussion

The settings for the CARD algorithm to obtain the results were following: at the beginning of the algorithm 50 rows of the membership matrix were chosen randomly to be the centers of clusters. The time constant τ was set to 10, the decay constant n_0 was set to 0.0002. A cluster was discarded if its cardinality (N_i) was less than 22.0.

When using URL similarity only, five clusters were generated. In the results obtained, cluster 0 is primarily referring to Dr. McGlinn's pages, cluster 12 is primarily referring to the CS pages and in some instances the cs200b web site, cluster 13 is referring to a combination of Dr. Wainer's 485 class, and Mr. Fong's cs311 and cs220 classes, cluster 25 is referring to a UNIX description page and cluster 34 is referring to primarily the cs220 pages. All of the clusters are valid clusters, in the sense that their intra-cluster similarity (D_{W_i}) is low and that inside the clusters the general URL access pattern is high.

The only cluster that seems questionable is cluster 25, where the URL access patterns are very low. Because of the unusually low URL access pattern this cluster could be characterized as a collection of unrelated pages, more like the leftovers from the other clusters.

When using URL and keyword similarity together, seven clusters are generated. It seems that the clusters from just using URL similarity have been broken down into more identifiable clusters when incorporating keyword similarity. For example when using URL similarity alone, cluster 0 is primarily composed of Dr. McGlinn's pages and all other clusters do not reference these pages. However when keyword similarity, clusters 1, 8, and 12 all reference Dr. McGlinn's pages.

The URLs that cause these three clusters to deviate, are the following: in cluster 1, the url /~mcglinn/Courses/cs202/McGlinnSlides/index.html is present, in cluster 8 the two urls /~mcglinn/home.html and /~mcglinn/index.html are present, and in cluster 12, the two urls ~mcglinn/Courses/cs202/JavaInformation/index.html and /~mcglinn/Courses/cs202/omework/index.html are present. Table 1 shows the URL access probabilities for the five previously mentioned urls in cluster 0 when using URL similarity only and clusters 1,8 and 12 when incorporating keyword similarity as well.

What appears to be happening is that when using keyword similarity the similarity of the URLs given in Table 1 is increasing, since these pages share a common theme with the other pages in Dr. McGlinn's web site.

This increased similarity gives these URLs a chance to compete when the clusters are being formed and lend enough similarity that they help delineate the one cluster referring to Dr. McGlinn's web site in Table 1 into three separate clusters in Table 2.

Another interesting development is that when using URL similarity the URLs for cs200b class were simply part of a cluster that primarily referred to several types of computer science pages in cluster 12, however when incorporating keyword similarity the cs200b class has been assigned its own cluster.

However, using keywords has rewards as well as problems. If we examine the results further, we see that when incorporating keyword similarity we often see low URL probabilities. Also, in cluster 0, the url probability for URLs index.html and ~cs200b-1/index.html the URL probability is reasonable, but the other URLs in the cluster do not have very high URL probability. One very possible reason for this is that we were unable to parse keywords from all documents. The session data was obtained in Fall 2004 but the CARD Algorithm was not fully tested until early Spring 2005. Therefore several web pages were already taken off the web, rendering them impossible to parse. In addition, several pages were password protected and due to time constraints we were not able to gather all of the passwords, therefore we could not obtain the HTML source for these pages for parsing. Therefore, these pages had no added keyword similarity. However, keep in mind that the URL similarity was factored in. Therefore in some instances, course web pages will still be found together, for instance, although Mr. Fong's 220 class was inaccessible for keywords, the pages were still clustered together.

TABLE I
URL ACCESS PROBABILITIES WHEN USING ONLY URL SIMILARITY AND WHEN INCORPORATING KEYWORD SIMILARITY

URLs	Cluster 0 P_{ij}	Clusters 1, 8, 12 P_{ij}
/~mcglinn/Courses/cs202/Slides/index.html	0.040	0.120 (cluster 1)
/~mcglinn/home.html	0.120	0.400 (cluster 8)
/~mcglinn/index.html	0.120	0.400 (cluster 8)
/~mcglinn/Courses/cs202/JavaInfo/index.html	0.030	0.250 (cluster 12)
/~mcglinn/Courses/cs202/homework/index.html	0.210	0.250 (cluster 12)

IV. PERFORMANCE IMPROVEMENT: THE SECOND MODIFICATION

A. *Parallel Implementation*

When clustering user sessions into profiles, the actual CARD algorithm is relatively fast. Most of the time spent in execution is during the early data preparation phase. For example (2) and (4), where the session similarity is computed, take the majority of time to execute. However (2) and (4) can be simply executed by using for loops. This combination of for loops and matrix mathematics make the algorithm easy to implement in parallel.

If we look closely at (2) and (4), we can observe that $s[i][j] = s[j][i]$. For example if we find the similarity between session 0 and all other sessions yielding the row $S[0][0..N_{url}-1]$ then on subsequent iterations we will not need to find any other session's similarity to session 0 because this has already been computed. In general when using (2) and (4) on a serial computation we only need an answer matrix S which has the following number of elements exhibited in (11).

$$numElements = N_s * \sum_{i=0}^{Ns-1}(N_s - i) \qquad (11)$$

Therefore when we compute (2) and (4) we only compare session 0 to sessions 1 through N_s, session 1 to sessions 2 through N_s and so on. In general with each new session we do one less comparison with the other sessions. If we could envision what the matrix looks like when we do this computation it could be illustrated by Fig. 3.

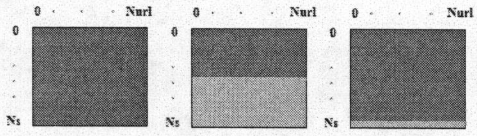

Fig.3 Computation during iterations 1, N_s /2 and N_s

In a parallel computation, where we will denote N_{proc} as the number of processors available, it would seem most appropiate that each processor should calculate the similarity of N_s/ N_{proc} number of sessions to all other sessions. For example if there are 1116 sessions and 9 processors then each processor will calculate the intersession similarity for 124 sessions. If the number of sessions does not evenly divide by the number of processors, then we add one session per processor, until all remaining sessions have been distributed.

To begin the parallel computation the root node in the parallel cluster reads in a matrix s of dimensions $N_s x\ N_{url}$. It then distributes to each processor a *beginID* and a *endID*. The *beginID* and *endID* are the ids of the sessions for which each processor will compute intersession similarity. For example if there are 1116 sessions and 9 processors, then processor 0 will compute intersession similarity for sessions 0-123, as will processor 1 for sessions 124-147 and so on until processor 8 will compute intersession similarity for sessions 992-1115. Once each processor has received its *beginID* and *endID*, then the root node will distribute the row *beginID* through the N_s*-1* row to each processor.

Before each processor can compute (2) and (4) it must allocate space in which to hold its intersession similarity. If we define N_{rows} as the number of sessions per processor then the number of elements for the matrix S per processor would be found in (12).

$$num\_matrixS\_elements = N_{rows} * \sum_{i=0}^{Nrows-1}(N_{rows} - i) \quad (12)$$

Once the answer matrix S has been formed, (2) can be computed. The modified equation, (2) for parallel will simply find intersession similarity for sessions beginID through endID as related to sessions beginID+1 through N_s. Similarly. a modified equation, (4) can be computed, this equation also finds intersession similarity for sessions beginID through endID as related to sessions beginID+1 through N_s

B. *Results and evaluation*

The computational complexity of finding the intersession similarity is $O(n^4)$. We will consider p to be the total number processing nodes available for the computation. Therefore, we can define the efficiency *(E)* as the time for the serial algorithm (referred to as T_c) divided by the product of the parallel time (T_p) and p [8]:

$$E = \frac{T_c}{pT_p} \quad (13)$$

And the speed up to be [8]:

$$S = \frac{T_c}{T_p} \quad (14)$$

An efficiency of 1 implies that the scalability of the system is unlimited as the problem size grows. Moreover not considering the main memory and cache limitation effects in a single computing node; speedup is bounded by a threshold value subject to Amdahl's law [8].

For implementation and evaluation of the concepts presented, a dedicated OSCAR cluster [9] of nine nodes was utilized. All compilations were done using the LAM MPI libraries, packaged with the OSCAR middleware. Each of the computational nodes consists of a single Pentium 4 class processor, with 786 megabytes of memory, a clock speed of 2.0 gigahertz and a 8 KB on board cache. The computer that was used for serial computation had a Pentium 4 class processor, with 512 megabytes of memory, a clock speed of 2.0 gigahertz and 512 KB on board cache.

The data that was used in testing ranged from 100 to 1116 user sessions. The serial and parallel runtimes are presented in Fig. 4. The speedup of the parallel implementation is illustrated in Fig. 5 and its efficiency is shown in Fig. 6.

The speedup and efficiency seem to reach a peak level at about 500 user sessions and then the speedup levels to about 4. In general we have sped up the algorithm by about 3.5 times. Although this may seem modest, if we look at Fig. 4 we can see that with 1116 user sessions it originally took 4.4 hours to find the session similarity when using serial computation and when using parallel computation we have decreased this time to 1.25 hours.

The suggested use of this parallel computation is to first use the serial algorithm to find the similarity of all URLs to each other, and the occurrence of each URL in each session. This would give us two input files, *urlSimilarity.txt*, which would contain a matrix of size N_{url} x N_{url}, and *urlOccurrence.txt*, which would also contain a matrix of size N_s x N_{url}. These files could then be sent to the OSCAR cluster and the session similarity could then be computed on the cluster. After the session similarity has been computed, the cluster will output a file called sessionVsSessionSimilarity.txt which would be size N_s x N_s. This file could then be fed back into the serial computation so that the CARD algorithm can complete.

One relatively simple way to speed up the parallel computation further is to reduce the times that matrices are being sent from the root node to the other nodes. In general in the computation discussed, when the root node sends the similarity matrix to each matrix, this could be stopped, and instead each matrix could read in its portion of the similarity matrix from a file. In this way the time to send the matrix is completely diminished and all that is left is file access time and data retrieval from the file, which is quite fast when using the C programming language. However, this implementation was not performed due to time constraints.

V. CONCLUSION AND FUTURE WORK

Clustering algorithms have become an invaluable way to form user sessions into profiles. Once these profiles have been formed, then a recommendation engine can use these profiles to give user's suggestions on which URLs will be of interest to them. This can not only be useful, but profitable. For example a company who is selling items can use the profiles to make suggestions to a user about other items that they are interested in. Improvements in clustering are being made very quickly and these algorithms are becoming faster and more reliable.

In Section 2 we have shown that incorporating keyword similarity into URL similarity creates some clusters which have greater intra-cluster similarity. These clusters also show high URL probability rates. However, when incorporating keywords we have also seen that results are not perfect and that some clusters are formed with low intra-cluster similarity. A probable reason for this discrepancy is HTML documents that are offline and therefore cannot be parsed for keywords. In Section 3 we have shown that there can be a large speedup in overall computation when we find similarity using parallel computing.

Suggested improvements are to test modifications 1 and 2 on larger data sets. Another improvement would be to change the dataset from a computer science web site to a larger and more diverse website. Also, for the sake of using keywords, one should make sure that the majority of referenced HTML documents are online and if password protected all passwords are known so that the HTML source can be retrieved. In this way, also we think that a more diverse, complete and/or a larger dataset would demonstrate a more marked improvement due to the modifications. Also, when finding updated similarity the keyword similarity is given a weight of 2/3 and the URL similarity is given a weight of 1/3. It would be interesting to modify these weights and find the weights that give the best clustering outcome.

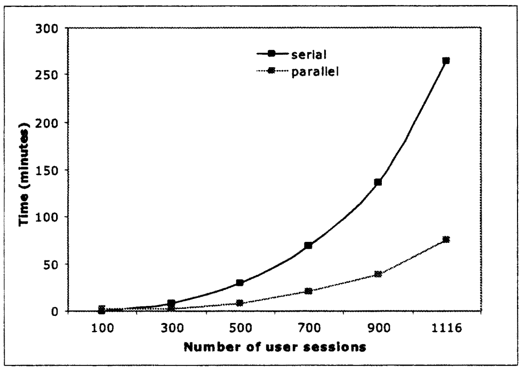

Fig. 4 Serial and parallel run times when finding session similarity

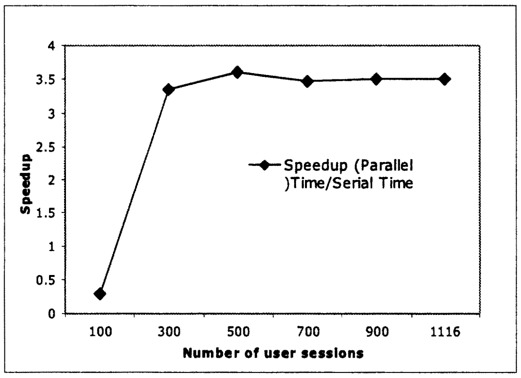

Fig. 5 Speedup Graph

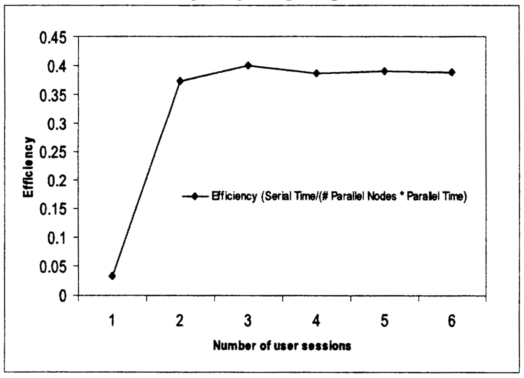

Fig. 6 Efficiency Graph

References

[1] O. Nasroui, R. Krishnapuram et al. "Extracting web user profiles using relational competitive fuzzy clustering" International Journal on Artificial Intelligence Tools, Vol. 9, No. 4, 2000, pp. 509-526.

[2] O. Nasraoui, C. Petenes, "An intelligent web recommendation engine based on fuzzy approximate reasoning" 12th IEEE International Conference on Fuzzy Systems, Vol. 2, May 2003, pp. 1116 – 1121.

[3] R. Krishnapuram, A. Joshi et al, "Low-complexity fuzzy relational clustering algorithms for Web mining," IEEE Transactions on Fuzzy Systems, Vol. 9, No. 4, Aug. 2001, pp. 595-607.

[4] M. Vazirgiannis, "Data Mining: Concepts and Techniques," Tutorial Paper, Proc. ADBIS 2001, Vilnius, Lithuania.

[5] F. Heylighen, "Collaborative filtering," website: http://pespmc1.vub.ac.be/COLLFILT.html, March 2001.

[6] G. Linden, B. Smith, and J. York, "Amazon.com Recommendations: item-to-item collaborative filtering", IEEE Internet Computing, Vol. 7, No. 1, Jan/Feb 2003, pp. 76-80

[7] M. F. Porter, "The Porter Stemming Algorithm," Program, Vol. 14, No. 3, 1980, pp. 130-137.

[8] A. Grama, Introduction to parallel computing 2nd edition, Addison Wesley, 2003.

[9] "OSCAR (Open source cluster application resources)," website: http://oscar.openclustergroup.org.