The Personalized Services in CADAL Digital Library *

Yin Zhang
the College of Computer Science
ZheJiang University,
HangZhou, China
zhangyin98@zju.edu.cn

Jiangqin Wu
the College of Computer Science
ZheJiang University,
HangZhou, China
wujq@zju.edu.cn

Yueting Zhuang
the College of Computer Science
ZheJiang University,
HangZhou, China
yzhuang@zju.edu.cn

Cheng Ma
ZheJiang University,
HangZhou, China
mach416@gmail.com

Chuan Yuan
ZheJiang University,
HangZhou, China
yc8244@163.com

Chunhe Wang
ZheJiang University,
HangZhou, China
chunhezju@yahoo.com.cn

ABSTRACT

CADAL is a great digital library project of digitizing one million digital books and publishing them to the internet users. It's obvious that users confront with the information overload problem when visiting the CADAL portal. Therefore, we have been concerned with providing useful and flexible personalization services to reduce the users' time and energy cost of finding interesting information. We have a extensible framework for personalization services in CADAL, including frontend UI and backend module. Since the log data is plentiful and easily recorded, it's our initial step to construct the recommender system based on the massive log data. The approach implemented by us is based on two kinds of data structure: red-black header tree and prefix subtree. The results of experiments on real-world log data confirm the efficiency and excellent scalability of our approach with the large number of items and sessions.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms, Experimentation

Keywords

Personalization, Web Usage Mining, Sequence Mining, Recommender, Web Logs

*This work is supported by National Natural Science Foundation of China (No.60525108,No.60533090), China-American Digital Academic Library project funded by State Development and Reform Committee under Grant No.1659[2004] , and Program for Changjiang Scholars and Innovative Research Team in University (IRT0652).

1. INTRODUCTION

The goal of CADAL(China-American Digital Academic Library) project is to build a large scale digital collection of one million books, which are accessible to everyone over the Internet. CADAL was funded by China Education Ministry and State Development and Reform Committee, and was also part of Universal Digital Library. Now the number of digitized book is above one million, which is a great step to store all literatures about Chinese traditional civilization and modern science, covering every aspects of humankind knowledge. All the digitized contents have been transferred to the Library of Zhejiang University, in which the project administrative center locates. The College of Computer Science in Zhejiang University has undertaken the development of key technologies sustaining the portal of CADAL (http://www.cadal.zju.edu.cn) [9, 10] since 2003, which aim to make it easy for users over Internet to search or browse what they are interested in explicitly or implicitly.

With the large increase of the number of digitized book available to users, it is imperative to provide the personalized services to reduce the problem of information overload for users during the development of the portal. Up to now, rule and mining approach have been designed, implemented and delivered to serve the users. In the remainder of this paper, the architecture of personalized services in CADAL portal is to be described in section 2, which briefly covers the user interface based on web 2.0 techniques, the rule approach and recommendation based on mining techniques. In section 3 is how to mine the frequent sequential access pattern and then predict the recommendation. The results of experiments of mining approach proposed by us is described in Section 4.We draw a conclusion and make a future work plan in Section 5.

2. THE ARCHITECTURE OF PERSONALIZED SERVICES

Figure 1 shows the architecture of personalized services in CADAL portal, which are partitioned into frontend and backend. The users interact with portal through frontend i.e Personalized Index Page, Reader Page, Setting Page. The individual reads the content of book through Reader page, at the same time the logging module of portal records all book pages read by him/her. Reader page also provides

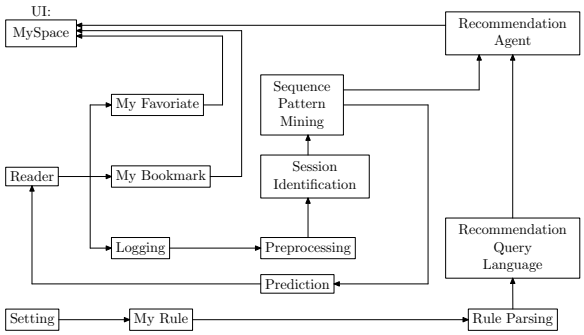

Figure 1: The current architecture of CADAL Personalized Services

functionality to make it easy for user to bookmark the book pages of interest and add current book to his personal collection. Moreover, one can set the personalized rules through Setting page, which are regarded as one's preference. The backend will parse all the inputted rules to generate constraint set, which is the input of recommendation agent to match and filter out the related books. The preprocessing, transaction identification and sequence pattern mining module are applied on the logs recorded by logging module to discover the frequent sequential access pattern and then do the recommendation.

It is our objective to reduce as much time and effort cost as possible for users to get what they are interested in. Thus we have exploited Ajax 2.0 techniques to put all personal and filtered information related to individual user into the Personalized Index page, through which one can quickly outlines all aspects of novel information added into Digital Library recently and implicit knowledge discovered by background mining algorithm. The web personalized index page (see figure 2) is partitioned into many square areas, which can be classified into four category. One category is responsible to "my favorite book" chosen by user itself. Another category is "my bookmark" added by user itself when reading some books. The third category is the list of filtered books by portal system according to personalized rule supplied by user itself. The last category is the list of books recommended by the frequent sequence pattern mining algorithm applied on accumulated user browsing logs. The details of personalized rule and mining algorithm are to be discussed in later sections.

3. RECOMMENDATION BASED ON SEQUENTIAL PATTERN MINING

3.1 Motivation

In the past decade, web usage mining defined by [1] has become an important research area, which focuses on predicting and learning users' preferences on the internet. Most of the studies try to discover all association rules prior to generating recommendations, while we are more interested in real-time dynamic recommendation. Several studies using tree structure have been proposed. [2] proposed a frequent sequences tree (FS-tree) structure for mining the web users' behavior over time. [3] proposed a WAP-tree for mining task. The WAP-tree is similar to the FP-tree proposed by Han et al. [5].

Figure 2: The entry page of CADAL Personalized Space

In CADAL portal, we initially adopted a special process to transform the log data to a condensed navigational pattern tree proposed by [6]. Soon we found the problem that original NP-tree algorithms didn't take the large number of items into account, but there are one million books in CADAL with which previous NP-tree algorithms don't scale up. Therefore we utilized the red-black header tree [7, 8] instead of the previous header link list, then the construction time of NP-tree is greatly shortened. Furthermore, we discovered that RB-tree's $log(n)$ insert and search cost at worst case can also contribute to the improvement of performance of finding the first item in the session being processed and generating the recommendation. Finally we compute the lift [4], rather than the traditional support, in order to generate a few more appropriate recommendations.

3.2 Problem Statement

Let the set of available books be $B = \{B_1, B_2, \cdots, B_n\}$, and $T = \{(1, B_i), (2, B_j), \cdots, (n, B_k)\}$ is an sequence of books accessed during user's single visit to portal ,where the item (i, B_k) represents the kth book was accessed the ith time within a reading session T. In our current system implementation, preprocess module identifies the log lines related to book access in the raw log files, with irrelevant items removed, e.g. gif, jpg. Session identification module divides book access sequence into different sessions if the time between book accesses exceeds the default timeout limit of 30 minutes. Thus all the identified sessions make up the reading trail database $RDB = \{t_1, t_2, \cdots, t_l\}$.

We denote n-sequential pattern as a historical sequential pattern P containing n items. The occurrence frequency of a sequential pattern P is the number of sessions containing P in RDB. P is a frequent sequential pattern if P's lift satisfies a predefined minimum lift threshold, at least 1. In a word, we are concerned with predicting the next $|P_a|+1$th book according to the active user's historical reading pattern

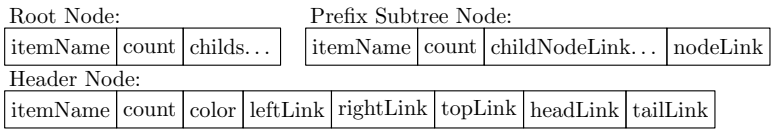

Figure 3: The format of three kinds of node

P_a, given a reading trail database RDB, and a minimum lift threshold of 1. The details of our approach are described in the next sections.

3.3 Definition of Navigational Pattern Tree with Red-Black Header Tree

We utilized data structure and related algorithms of navigational pattern tree to efficiently predict and recommend the $n+1$th appropriate book to the user. A navigational pattern tree consists of three parts shown in figure 3: a *root node*, a *header tree* and a array of *prefix subtree* which is the child of the root node.

The *root node* of a navigation pattern tree is made up of three elements: *itemName*, *count* and *childNodeLink*, where *itemName* is set to be null, *count* stores the number of sequential sessions in the navigation pattern tree, *childNodeLink* points to a array of prefix subtree.

A *prefix subtree node* has four parts: *itemName*, *count*, *childNodeLink* and *nodeLink*, where *itemName* is the ID of the accessed book, *count* is the accessed number of portion of the path from root node to this node, *childNodeLink* points to another prefix subtree node that represents subsequently accessed book within the same session, *nodeLink* links to other prefix subtree node carrying the same book ID in the navigational pattern tree.

Each node in the *header tree* contains eight fields: *itemName*, *count*, *leftNodeLink*, *rightNodeLink*, *color*, *topSubtreeNodeLink*, *headSubtreeNodeLink* and *tailSubtreeNodeLink*. The field *itemName* registers the book ID. The total access number of the book in the navigational pattern tree is stored in the field *count*. Since we construct one kind of binary balanced tree to store header nodes,*leftNodeLink* links to header node carrying smaller book id, *rightNodeLink* points to header node carrying bigger book id. When the red-black tree is constructed, *color* field is set to red or black accordingly. the insert and search time in red-black tree is proportional to $log(|book|)$, which is a great speedup with respect to original link list implementation. That was proved by the results of experiments in section 4. *topSubtreeNodeLink* is used to find the prefix subtree whose first node carrying the same book id, which reduce significantly the comparison cost when constructing the sequential pattern tree because searching time in one million books is reduced greatly. *headSubtreeNodeLink* links to the first node carrying the same book id in the navigation pattern tree.*tailSubtreeNodeLink* points to the last node carrying the same book id in the navigation pattern tree.

Figure 4 is an example of constructed navigational pattern tree at certain time.

3.4 Navigational Pattern Tree Construction

we proposed a set of routines to construct a navigational pattern tree defined in previous section. The algorithm of constructing a navigational pattern tree consists of four seamlessly related routines: *SPTreeConstruction*, *TopPrefixSubTreeNode*,*NextPrefixSubTreeNode* and *ConnectHeaderLinks*. The whole process of constructing a navigation pattern tree is controlled in *SPTreeConstruction*. At the beginning, *SPTreeConstruction* reads one session from the RDB, then extract the first item from this one, call *TopPrefixSubtreeNode* to do two things, one is to adjust header tree, the other is to decide whether to create a new prefix subtree or reuse the existing one. Next, on the rest of items within the session is applied *NextPrefixSubtreeNode*, which takes charge of the construction of prefix subtree and calls *ConnectHeaderLinks* to adjust links from header node to the respective subtree node. The detail of each routine is described as follow.

SPTreeConstruction accepts the trail database RDB as the input, returns the Navigation Pattern Tree:

1. Create the Root Node. Initialize itemName to null, set the count to 0.
2. Read the next available session T from RDB. Do step 3-6;
3. Pick the first item B_1 of T and then pass it to *TopPrefixSubtreeNode* to get the prefix subtree node *preSubtreeNode*. Create the link from RootNode to *preSubtreeNode*, Increase count of Root Node by 1. Set $i = 1$
4. $i = i + 1$; Get the next available item B_i; if B_i is null, goto Step 6
5. Pass the *preSubtreeNode* and B_i to *NextPrefixSubtreeNode*, which returns a prefix subtree node to update *preSubtreeNode*. goto Step 4.
6. goto Step 2;
7. After processing all transactions, return the navigational pattern tree containing Root Node, a set of Prefix Subtree and a Header Tree.

TopPrefixSubtreeNode accepts the item name as the input, returns the appropriate prefix subtree node.

1. search the header node carrying the same book id in the red-black header tree,if found, continue; otherwise goto step 5;

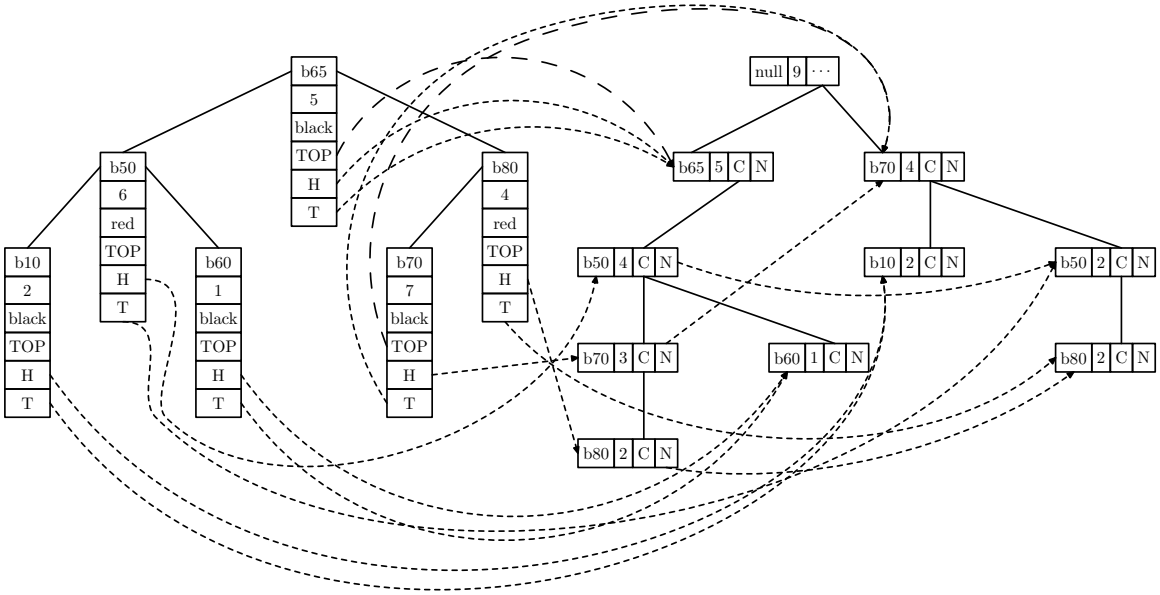

Figure 4: An example of navigational pattern tree

2 check whether its' *topSubtreeNodeLink* is null. if null, goto step 3, otherwise goto step 4;

3 create a new prefix subtree node *preSubtreeNode* carrying the same book id and make *topSubtreeNodeLink* link to it. Set the *count* in the new subtree node to 1. Return *preSubtreeNode*;

4 get the prefix subtree node referenced by *topSubtreeNodeLink*, increase the *count* of it by 1.Return *preSubtreeNode*;

5 insert new header node *hNode* carrying the same book id; make its' *topSubtreeNodeLink* and *headSubtreeNodeLink* link to a new created prefix subtree node *preSubtreeNode* carrying the same book id. Set the *count* of *hNode* and *preSubtreeNode* to 1. establish the link between *RootNode* and *preSubtreeNode*. In the end, return *preSubtreeNode*.

NextPrefixSubtreeNode below is used to handle the growth of the prefix subtree. *NextPrefixSubtreeNode* accepts the prefix subtree node $PreSubtreeNode$ and the book B_i as arguments, returns the appropriate prefix subtree node.

1 get the next available child node *child* of *PreSubtreeNode*; if existed, continue; otherwise, goto Step 3;

2 if the item name of *child* equals the ID of B_i, increase count of *PreSubtreeNode* by 1, return *child*; Otherwise goto step 1;

3 create a new node *preNode* as the child of *PreSubtreeNode*, then pass *preNode* to *ConnectHeaderLinks* to adjust links between *preNode*and the header node in header tree carrying the same book id. Return *preNode*.

The routine *ConnectHeaderLinks* accepts the prefix node $preNode$ as the argument;

1 search the red-black header tree for the book id carried by *preNode*. If found, continue, otherwise goto step 3.

2 name the matching header node *header*, get the prefix subtree node *prefixSubtreeNode* referenced by *tailSubtreeNodeLink* of *header*. Increase *count* of *header* by 1. Modify *nodeLink* of *prefixSubtreeNode* to point to *preNode*. Modify *tailSubtreeNodeLink* of *header* to link to *preNode*. Return;

3 create a new header node *nHeader* carrying the item name of *preNode*, which is inserted into the red-black header tree in at most $log|headertree|$ time. Both *tailSubtreeNodeLink* and *headSubtreeNodeLink* of *nHeader* are linked to *preNode*. Set *count* of *nheader* to 0. Return.

3.5 Recommendation Algorithm based on Navigational Pattern Tree

Given a navigational pattern tree *PTree*, a historical sequential pattern P_a of a active reader and a minimum lift threshold of 1, we bring our efforts to bear on predicting the next $|P_a| + 1$-*th* book to the reader.

The whole algorithm is composed of four routines: *Recommend, FindFirstCandidates, FindNextCandidates, GroupAndPrune*. *Recommend* is the main procedure controlling the whole process and manages the inputs and outputs of the other three routines. The whole working scene of the algorithm is described as follows: extract the first book B_1 of the historical sequential pattern P_a of active user, *FindFirstCandidates* generates the candidate set *cadidates* of prefix subtree nodes carrying the same book id of B_1. Then, get the next available book B_i of P_a, apply *FindNextCandidates*

to B_i and *candidates*, limit the search space among the child nodes of nodes in *candidates*, generate the candidate set of i-th level prefix subtree nodes carrying the same book id of B_i. In the end, select a subset of $|P_a|+1$-th level prefix subtree nodes as the recommendations, whose lift satisfies the minimum threshold of 1.

Recommend accepts the $PTree$, P_a as arguments,returns the candidate recommendation items delivered to the user:

1. extract the first item B_1 of pattern P_a,
2. pass B_1 to *FindFirstCandidates* to generate the candidate set *candidates* of prefix subtree nodes carrying the same book id of B_1. Set $i=0$;
3. Increase i by 1. check whether i is less than the length of the historical sequential pattern P_a. If yes, continue; otherwise goto Step 5.
4. pass the *candidates* and the ith item B_i of P_a to *FindNextCandidates* to update *candidates* carrying the same book id of B_i. goto Step 3.
5. pass *candidates* to *GroupAndPrune* which merges the subtree nodes carrying the same book id, prunes the infrequent items and returns the frequent ones as the final recommendation.

FindFirstCandidates accepts the B_1 as arguments, returns the candidate set of subtree nodes carrying the same book id of B_1:

1. search the header node *header* carrying the same book id of B_1 from the red-black balanced header tree. if found, continue; otherwise return;
2. get the subtree node *candidateNode* reference by *headSubtreeNodeLink* of *header*, add it to the candidate set *candidates*.
3. check whether *nodeLink* of *candidateNode* equals *tailSubtreeNodeLink* of *header*. if yes, return *candidates*, otherwise continue.
4. set *candidateNode* to the subtree node referenced by *nodeLink* of *candidateNode*. Add the new *candidateNode* to the *candidates*, goto Step 3.

FindNextCandidates accepts the *candidates* and B_i as arguments, returns the ith-level candidate set $nCandidates$:

1. get the next available subtree node *candidateNode* from *candidates*. If found, continue; otherwise return $nCandidates$.
2. get the next available child node *child* of the candidate nodes *candidateNode*. If found, continue; otherwise goto step 1;
3. check whether the book id of *child* equals the book id of B_i, if yes, add *child* to $nCandidates$, otherwise continue. goto Step 2.

Table 1: The example of reading trail database

Access Sequence	Occurrence
B65,B50,B70,B80	2
B65,B50	1
B65,B50,B70	1
B65,B50,B60	1
B70,B10	2
B70,B50,B80	2

GroupAndPrune accepts the candidate set *candidates* of prefix subtree node at level $|P_a|$ as arguments:

1. Initialize a map *Recommendation* containing zero elements. The $|P_a|+1$ level prefix subtree node is the key of map, the $|P_a|$ level prefix node is the value of map.
2. Get the next available subtree node *candidateNode* from *candidates*, if existed, continue; otherwise goto Step 5
3. Get the next available child node *child* of *candidateNode*, if existed, continue; otherwise goto Step 2.
4. Search for *child* in the set *Recommendation*. If there is a matching entry in the map, *child1* and *parent*, increase the *count* of *child1* by the *count* of *child*, increase the *count* of *parent* by the *count* of *candidateNode*. Otherwise, add *child* and *candidateNode* to *Recommendation* map. Goto Step 3.
5. Enumerate over the map, compute the respective lift value $\frac{child1.count}{parent.count} \times \frac{rootnode.count}{header.count}$, where the book id of *header* is equal to that of *child1*. In the end, return the set of keys *child1*, whose lift is above 1.

3.6 A Running Example

In this section, let me show a running example of NP-tree construction and predicting recommendation based on NP-tree. Table 1 is a recorded reading trail database, which contains 6 books accessed during 9 sessions. There are five types of access sequence listed in left column of table 1, respective right column is the number of occurrence. B65 denotes that a book id is 65, which is regarded as the sorting key in red-black header tree.

The RDB is recorded according to the time of occurrence of reading. Hence {B65,B50,B70,B80} occurred first, {B65,B50} occurred second, $\cdots$. When analyzing the RDB sequently, first encounter the session of {B65,B50,B70,B80}, looking for B65 in the header tree containing zero elements. Therefore insert header node of B65 into header tree, at the same time create a prefix subtree node of B65 as a child of root node and modify the top, head, tail pointers of header node of B65 to link to prefix subtree node of B65. The processing of B50, B70, B80 are like that of B65 except not modifying top pointer and linking them as a branch in the prefix

subtree, since they are not the first element of the reading session. The session of {B65, B50} is processed second, since B65 is in the header tree, increase the count in the header node of B65 and the prefix subtree node B65. So do processing B50. Next is the session of {B65, B50, B70}, the handling of it is similar to that of {B65,B50}. The handling of the session {B65, B50, B60} is a little different because the prefix subtree node B60 is another child of that of B50. So do handling the rest of reading sessions. After processing, we saw the result NP-tree with read-black header tree in Figure4.

Suppose that one read a book B70, first identify the header node of B70, use the head pointer of it to get the head prefix subtree node of B70 in the branch {B65, b50, B70, B80 }, then B80 is one candidate recommendation. Furthermore, alongside the nodeLink of former prefix subtree node of B70, another that of B70 is got, its child includes {B10, B50}. So the candidate recommendation set is {B80, B10, B50}. The criteria of ranking is the lift computed by $\frac{child.count}{parent.count} \times \frac{rootnode.count}{header.count}$. The lift of B80 is $\frac{2}{3} \times \frac{4}{9} = 1.5 > 1$, that of B10 is $\frac{2}{4} \times \frac{2}{9} = 2.25 > 1$, that of B50 is $\frac{2}{4} \times \frac{6}{9} = 0.75 < 1$. Therefore, {B10, B80} is the final recommendation list.

4. EXPERIMENTS

In this section, we report the experiments and respective analyses performed on the real-world log data collected during the 411 days from May,2006 to July,2007. All experiments are performed on a computer with a CPU clock rate of 3GHz(Intel Pentium 4 630) and 1GB of main memory. All programs are written in Sun Java 1.5, and they ran on the Microsoft Windows XP with SP2. Table 2 shows the statistics of each attributes of real-world log datasets collected by logging module of CADAL portal and the time of original and proposed method at NP-tree construction and recommendation.

1. *MonthNum*: The number of months within which the log data was recorded. For instance, 4 in the second row denote a time interval of 4 months, from May 11,2006 to September 10,2006.

2. *LogNum*: The number of log lines collected during the respective number of months. The format of log lines follows the W3C extended log file format, but just the log lines related to reading book are reserved, other useless lines,e.g. gif, jpg, are removed before experiments. The total number of all valid log lines are 2,056,207 shown in the last row, whose time range covers the 13.5 months.

3. *BookNum*: The number of books read by internet users. Up to July 1, 2007, 410,034 books were accessed, while just 81,499 books are read one year ago.

4. *SessionNum*: The number of sessions of reading sequence of books, which are segmented according to whether the value of time interval between reading books exceeds the default timeout value of 30 minutes. The average length of session is $\frac{2056207}{266858} = 7.7$,

5. *UserNum*: The number of users accessing books in CADAL. Here one user means one unique ip address.

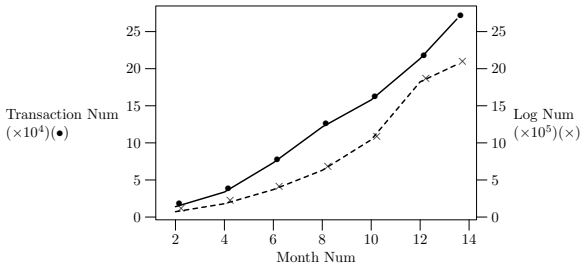

Figure 5: The plot of log line number and respective transaction number w.r.t month number

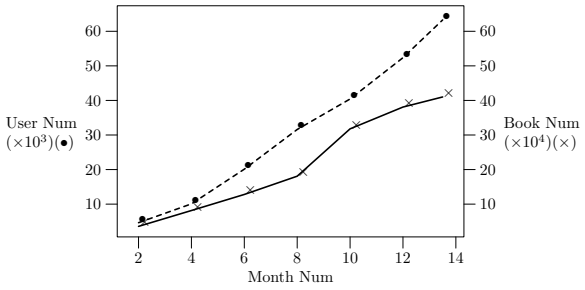

Figure 6: The plot of user number and book number w.r.t the month number

The increase rate of users is roughly consistent with the increase rate of books read, shown in figure 6.

The figure 5 shows that the change ratio of session number rose up steadily with respect to the month number before or after 8 months. Thus the lowest average number of books read by user is 180809/121456 = 5.723. While the slope of the number of log lines rose up steadily during 12 months, the slope of the most recent 1.5 months dropped off a little, approximately equal to the slop of from 6 to 10 months. The steadily increasing number of log lines indicates that the more people accessed the portal,the more books were read.

The figure 6 shows that the slop of user number line is completely consistent with that of transaction number in figure 5 accordingly, which corresponds to our transaction identification rule of grouping transactions according to the user ip address and timeout value. The trend of the number of books accessed over time frame is one of steady, stable growth. Up to July 1, 2007, users have accessed 410,034 unique books, which are just 41 percent of one million books because only about 300,000 ancient and public domain digital books have been available to everyone, other copyrighted books have been available to a few privileged persons.

The figure 7 demonstrates the total construction time of navigation pattern tree with respect to the different dataset over the different time frame, using original method [6] and our proposed method. The total construction time depends on two factors: one is the number of sessions in dataset, the other is the algorithm of construction. From the figure 7 we can see that the total construction time of proposed method increase steadily as the number of sessions climbs. When

Table 2: The manifold statistics of logs and time of original and proposed algorithms over time frame

MonthNum	LogNum	BookNum	SessionNum	UserNum	NPTTime(s)	OrgNPTTime(m)	RecomTime(s)	OrgRecomTime(m)
2	72257	35631	13964	4625	8	1.9	18	1.2
4	179555	81499	33686	10072	28	9.4	24	4.2
6	370134	127970	73226	20111	8	25	103	8.5
8	630127	180809	121456	31592	165	52.6	110	40.75
10	1042008	317801	157922	40415	239	132.6	260	80.2
12	1822763	380832	213560	52358	393	281.2	227	137
13.5	2056207	410034	266858	63280	550	332	267	189

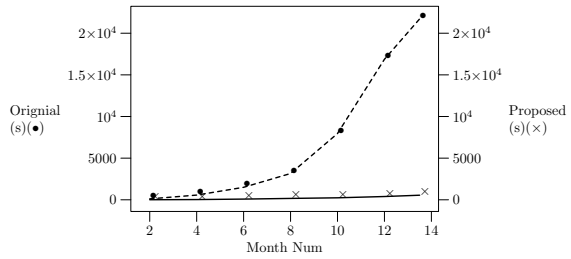

Figure 7: Time of building navigational pattern tree w.r.t the month number. original method VS proposed method

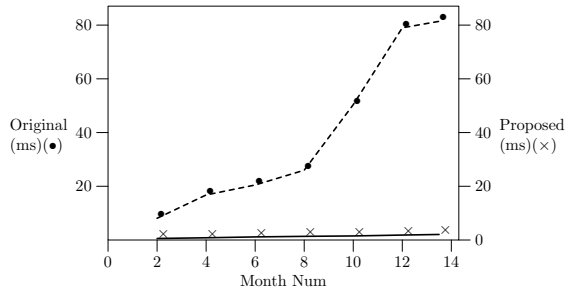

Figure 8: Average time of processing one session w.r.t the month number. original method VS proposed method

enumerating the sessions of first 2 month, the total time is just 8 seconds. Even with the all 266,858 sessions over 13.5 months, the scalability of our algorithm is high, the total time of NP tree construction is 550 seconds. But the construction time using original method will increase in a order of magnitude. The construction time over all log dataset is about six hours. Furthermore, From the figure 8, we can see clearly that original method quickly performs worse with the increase of number of sessions. While the average time per session of proposed method increases mildly, just from 0.57 to 2.06 milliseconds.

The figure 9 demonstrates the stable time cost of our recommendation algorithm comparable to original method. The real line shows the total recommendation time of proposed method that are applied to all sessions in the dataset to generate recommendation. The original method needs 3 hours to complete all recommendation process, while proposed method just needs about 4 minutes to do so. The average recommendation time per session is calculated by the formula $\frac{RecomTime}{SessionNum}$. The average time of original method fast climbs with the increase of session number. However,

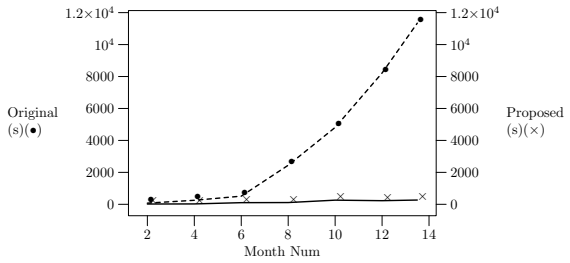

Figure 9: Time of prediction for all historical sessions w.r.t the month number. original method VS proposed method

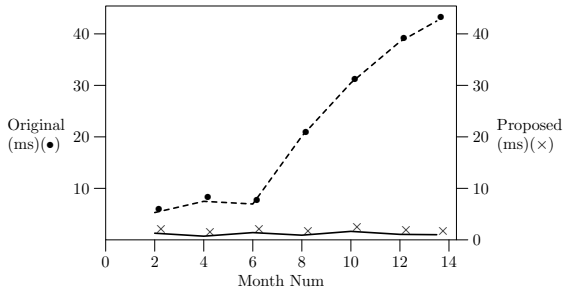

Figure 10: Average time of prediction for one historical session w.r.t the month number. original method VS proposed method

from the zigzag line of our method in figure 10, we can see that the average recommendation time per session of our method is independent of the total number of sessions, one recommendation according to the historical access sequence just costs more or less 1 milliseconds.

5. CONCLUSIONS

CADAL has been a important digital library, many users and books have been connected through our portal. To achieve the goal of reducing the users' time and energy cost of finding valuable information in CADAL, we have provided rule-based and mining-based personalized recommendation services. In order to overcome the difficulty brought by the large number of books, we proposed a new data structure of NP-tree with red-black header tree and related algorithms. The results of experiments proved the scalability and efficiency of proposed approach. In the future, we plan to provide a few personalization services based on the multidimensional hierarchical information, e.g. taxonomy of books, locations of users etc.

6. REFERENCES

[1] R. Cooley, B. Mobasher, and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *ICTAI*, pages 558–567, 1997.

[2] M. El-Sayed, C. Ruiz, and E. A. Rundensteiner. Fs-miner: efficient and incremental mining of frequent sequence patterns in web logs. In A. H. F. Laender, D. Lee, and M. Ronthaler, editors, *WIDM*, pages 128–135. ACM, 2004.

[3] C. I. Ezeife and Y. Lu. Mining web log sequential patterns with position coded pre-order linked wap-tree. *Data Min. Knowl. Discov.*, 10(1):5–38, 2005.

[4] L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *ACM Comput. Surv.*, 38(3), 2006.

[5] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[6] Y.-M. Huang, Y.-H. Kuo, J.-N. Chen, and Y.-L. Jeng. Np-miner: A real-time recommendation algorithm by using web usage mining. *Knowl.-Based Syst.*, 19(4):272–286, 2006.

[7] D. Knuth. *The art of computer programming, volume 3: sorting and searching*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1998.

[8] M. Weiss. *Data structures and algorithm analysis in Java,2/E*. Addison Wesley Publishing Co., Inc. Redwood City, CA, USA, 2007.

[9] J. WU, Y. Zhuang, and Y. Pan. Technical features in the Portal to CADAL. *Journal of Zhejiang University SCIENCE*, 6(11):1249–1257, 2005.

[10] H. Zhang, Y. Zhuang, J. Wu, and F. Wu. Research on grid-aware mechanisms and issues for cadal project. In E. A. Fox, E. J. Neuhold, P. Premsmit, and V. Wuwongse, editors, *ICADL*, volume 3815 of *Lecture Notes in Computer Science*, pages 489–490. Springer, 2005.