Predicting End-Users' Behaviors by Applying Microprocessor Branch Prediction Algorithms to Session Logs of a Networked Community

Kibum Kim Mary Beth Rosson
Department of Computer Science, Virginia Tech
Blacksburg, VA 24060, USA
kikim@vt.edu, rosson@vt.edu

ABSTRACT

If people can predict the future, they can organize their schedules more efficiently and complete their work in a more timely fashion. Many software applications try to assist users by utilizing some kind of prediction. For example, knowledge management systems provide various personalized recommendation tools that analyze each end-user's past behaviors and, based upon this analysis, predict their next actions. In the research domain for computer architecture, many architects have tried to develop accurate prediction algorithms in order to increase the performance of microprocessors. Adopting microprocessor branch predictions for the Web will raise many compelling, and sometimes problematical, issues. This research paper suggests both a new algorithm that uses session log analysis for predicting end-users' interaction with the Web and a new paradigm for decreasing perceptible latency by applying microprocessor branch prediction algorithms to session logs of a networked community.

Keywords: Prediction Algorithms, Web Personalization, Web Logs, and End-Users' Behaviors.

1. INTRODUCTION

A possible area in which we can apply a prediction algorithm is that of Web software. One of the most pervasive complaints regarding use of the World Wide Web involves the latency problem that an end-user usually feels. In an attempt to avoid sitting lethargically in front of a monitor while waiting for requested documents to show up, end-users have begun to pursue high-speed connection lines such as cable, T1, and T3. However, even with these advanced forms of connections, there is still a delay that all end-users experience for the simplest of reasons: documents will not start to be downloaded until after the end-user's request. If some "intelligent" algorithms can prefetch the requested dynamic contents before a user requests them, then this all-too-perceptible latency problem will be resolved.

Moreover, with respect to EUP (end-user programming), a Web prediction system is a good application domain for PBE (programming by example) because of its high repetition character. PBE has been used in desktop application domains before, but now it seems obvious that widely ranged Web end-users also receive benefits from the systems. Web applications would be reinforced greatly by intelligent support from a PBE component. The current algorithms generally used for predicting the next requested Web pages show, at most, 62.5% accuracy [4]. To increase the accuracy of prediction, we suggest adoption of the highly accurate microprocessor branch prediction algorithms presented in this paper.

Many methods for enhancing Web performance have come from other research fields in computer science. For example, a commonly used Web caching conception derives from file management in OS. The current main stream of Web prediction algorithms, known as Markov models [1], stems from data compression in Encoding. Some researchers have tried to apply a compiler optimization algorithm for the Web. In fact, point and path profiles are well known elements of algorithms for compiler optimization and Schechter, Murali, and Smith [12] have adapted these techniques for HTTP request prediction. This paper studies various prediction algorithms currently in use and suggests a new approach for increasing the efficiency of Web usability.

2. BACKGROUND

One of the most active and exciting research areas regarding the WWW involves the use of Web caching and prefetching to improve access time. Web caching has become popular because it is useful for reducing end-user perceived Web-site delays, network bandwidth usage, and server loads. By effectively utilizing existing resources, caching makes the Web appear faster and less costly. However, several recent studies indicate that the maximum hit rate achievable by any caching algorithm is just 40% to 50% [5]. In fact, one out of every two pages will not be found in the cache. This low hit rate occurs because most end-users browse and navigate the Web looking for new information. So caching old documents has limitations with regard to satisfying end-users' new requests.

The Web caching hit rate can be enhanced by combining it with prediction techniques to anticipate users' future requests and to preload new pages. The predicted pages will be prefetched into a client-side cache, even though there may not be an explicit end-user request. This transfer is virtually invisible: it can be accomplished as a background process while the end-user is busy with a current page. When the end-user requests the new link, the client-side cache will permit the requested page to be retrieved quickly. Based on other new requests, a server will continue to prefetch other new predicted pages and store them in the client-side cache. If we can exactly predict the pages an end-user will visit, and if those pages can be prefetched to a cache before they are requested, then the end-user will not perceive any latency. The requested pages will be provided quickly from the local disk without being transmitted through the Internet from a distant Web server.

By using path profiles in the session logs, we can predict an end-user's request behavior and generate dynamic pages before they are requested. If prediction is accurate and the requested pages are pre-generated by the server, the end-user will experience significantly lower latency periods regarding requests. However, existing prediction algorithms cannot provide perfect accuracy. The widely used current path prediction based on the Markov algorithm shows an accuracy rate of only 62.5% [4].

In contrast, due to the advanced research history of computer architecture, the average microprocessor branch prediction accuracy of known schemes achieves 94.4 % accuracy, and some current algorithms are claimed to achieve 97 % accuracy. There is much similarity between predicting branch results and guessing users' Web surfing habits. If we can apply the branch prediction algorithm in deciding end-users' anticipated behaviors, Web browsers can more effectively support the needs of end-users. In addition, this new prediction approach could also be used for other application domains,

such as personalized recommendation tools, search engines, and end-user centered Web designs.

3. RELATED WORK

Before posing its own solution to the problem of increasing Web usability, this study considers prior work done in the field.

3.1. Logged data can be used for various purposes

Web logs are usually organized with an IP address, date, time, size, requested URL, and parameter field. This logged data can provide vast information about Internet end-users, and thus it can be utilized to predict their future requests. Typically, logged data can be used in three ways [6]:

- First, the study of logged data can provide an understanding of how the software system is operating [7].
- Second, by examining quantitative records of end-users' repetition, researchers can identify interface problems [13]. For example, since a logged history shows that most users go to page C after visiting from page A to page B, a web interface designer might add a direct link to page C on page A.
- Lastly, data logs can be used to profile in chronological order end-users' surfing paths and interaction. This profiled access pattern can be used to predict which path the end-user most frequently follows.

Knowing the past tells us about the future. We can glean profile data from the client side or server side. The client side's log file records the path through the entire WWW visited by a specific client, while the server side's log file records the paths of all users who visit its site. In the case of a networked community, not only personal data logs, but also group data logs can be used to predict paths with high accuracy. Many other people in the group might already have followed the same path that the current user is traveling.

3.2. Prediction can be used for various applications

By analyzing a server's session logs, we can predict an end-user's possible surfing path among many links. Accurately prediction of an end-user's surfing patterns on the Web could eventually lead to a number of improvements in end-users' interactions. Some possible applications of predictive models include [10]:

- **Searching:** For example, the Google search engines use agents designed to gather information from the Web via random walks over each link [2]. With higher predictive accuracy in surfing, we can expect better search engines, because agents will reveal more realistic end-user surfing patterns.
- **Personalized Recommendation Tools for Knowledge Management**: By analyzing client side proxy server logs, a knowledge management tool can recommend related pages for each end-user. WebPersonalizer is a system that provides automatic personalization based on Web usage mining [9].
- **Web Site Design**: If we can predict end-users' flow through Web sites, then we can improve overall Web usability [3]. Web site designers can more efficiently arrange links to promote the desired flow of surfing through content. By gaining easy access to select materials, the end-user can avoid having to navigate through vast amounts of often poorly structured Web information.
- **Latency Reduction**: When an end-user requests a certain page, the server can prefetch several related candidate pages by prediction, based on the match of patterns in logged data. Roughly, the idea is that if a system could predict the content an end-user intends to visit next, then the system could prefetch that content. While the end-user processes one page of content, other pages could be prefetched from high-latency remote sites into low-latency local storage [10]. Web caching further reduces the response time.

4. CURRENT ALGORITHMS USED FOR PREDICTING END-USERS' SURFING PATHS

There are several approaches currently used for predicting the surfing paths that might be taken by end-users, among them the naïve approach, the Top-10 algorithm, the Markov algorithm, and point and path profiles.

4.1. Naïve approach

This approach prefetches most or all of the pages embedded in the currently accessed page. This approach does not consider the history of end-users' access patterns. It is a simple and straightforward algorithm because an end-user will generally follow one of the links accessible from the page currently being visited. The effectiveness of this approach depends on the average number of embedded links and the size of the bytes which these links represent. According to a recent experiment, the average number of links per page is 22.6, and the average size of these links is 7760 bytes [4]. Also, because of commercial portal sites that are bundled into links, the fraction of links and bytes accessed from the prefetched page is very low. The result of this experiment verifies the need for sophisticated prediction algorithms that will improve WWW performance.

4.2. Top 10 algorithm

This algorithm captures a well-known rule on the Web: popular documents are *very* popular. Markatos and Chronaki [5] recommend prefetching only a server's most popular documents. This approach combines the server's active knowledge of their most popular links with a client's logged profile. Actually, this idea has been used in the entertainment business to great success. Many music and video shops provide a section for the weekly Top 10 items that have been sold or rented. Based on the concept that many customers will look for products identified by such a list, merchants will stock the sections in question for easy access.

In the Top 10 approach, the server periodically calculates a list of its most popular documents and services them to its clients. To avoid enormous network traffic, this algorithm uses two dynamic threshold values. First, only the client who has made a sufficient number of requests (> *ACCESS_THRESHOLD*) to the server will receive prefetched documents. In other words, frequents clients are distinguished from occasional clients. Second, this algorithm dynamically limits the maximum number of documents (< *RESOURCE_THRESHOLD*) that can be prefetched from any server during any time session.

Another important consideration in the Top-10 approach involves how often the server should calculate the Top 10. If a new Top10 list is released too frequently, it would create a significant overhead for the server as well as for clients because under this system a client could potentially prefetch the same Top 10 documents every few minutes. On the other hand, if a new Top 10 is released only every few months, then it will probably not be credible, because those documents may either be already fetched by clients or considered out-of-date. Trace-driven simulation results show that one to two weeks is a reasonable interval to wait before releasing a new Top-10 [5].

Contrary to the previously proposed sophisticated prediction algorithms, the Top 10 approach has the advantage of being simple and easy to calculate, because prefetching the most popular

documents is a basic but effective heuristic. Although in most cases the algorithm can predict more than 40% of an end-user's requests, its performance requires some improvement before it can be truly competitive or effective.

4.3. Markov algorithm

Some early work in reference predictions has used the Markov algorithm [1]. A discrete Markov chain model can be defined as the tuple **<S, A, λ>** where **S** means the state space, **A** represents the transition probabilities matrix from one state to another, and **λ** is the initial probability distribution of the states in **S**. The fundamental property of Markov is that it predicts the next state by depending on prior states. For example, if the vector **S[t]** denotes the probability vector for all the states in time **t**, then:

$$\check{S}(t) = \check{S}(t\text{-}1)\, A$$

If the Markov chain contains **n** states, then the matrix of translation probabilities **A** is of size **n** x **n**. By using Markov chains, Web link sequence modeling can be created. The matrix **A** can be estimated in various ways. One general way is that the maximum likelihood principle is applied to estimate **A** and **λ**. Each element of the matrix **A[s, s']** can be estimated as follows:

$$A(s, s') = C(s, s') \,/\, \Sigma_{s''} C(s, s'')$$

$$\lambda(s) = C(s) \,/\, \Sigma_{s'} C(s')$$

The count **C(s, s')** represents how many times **s'** follows **s** in the training data. We use the transition matrix to estimate short-term link predictions. An element of the matrix **A**, say **A[s, s']**, can be interpreted as transition probabilities from state **s** to **s'** in one step. Similarly, each element of **A** * **A** will be interpreted as the probability of transitioning from one state to another in two steps, and so on.

With the end-user's logged path history **L(t-k), L(t-k+1)…..L(t-1)**, we can represent each path as a vector with a probability i at that state for that time, denoted by **i(t-k), i(t-k+1)…. i(t-1)**. The Markov chain model's estimation of the probability of being in a state at time **t** is shown as

$$\check{S}(t) = \hat{\imath}(t\text{-}1)A$$

Variants of the Markov chain models can be proposed to accommodate the weighting of more than one history state for path prediction [11]. In the following equations, each of the previous links is used to predict the future links, and they can be combined in various ways:

$$\check{S}(t) = a_0\hat{\imath}(t\text{-}1)A + a_1\hat{\imath}(t\text{-}2)A^2 + a_2\hat{\imath}(t\text{-}3)A^3 \ldots..$$

$$\check{S}(t) = \mathrm{Max}(\, a_0\hat{\imath}(t\text{-}1)A,\ a_1\hat{\imath}(t\text{-}2)A^2,\ a_2\hat{\imath}(t\text{-}3)A^3 \ldots..)$$

In practice, in order to select a list of probable paths that a user will choose, the state probability vector **S(t)** can be normalized and thresholded [11].

However, there are arguments that to a large degree that because the Markov prediction algorithm is too general, it discovers patterns that because they are already obvious in the HTML of recently accessed pages might be more simply extracted from there [4].

4.1. Point and path profiles

Point and path profiles are well known algorithms for compiler optimization. These techniques are adapted by Schechter, et. al, for HTTP request prediction [12]. For point profiles, the frequency for any unit path from page X to Y is considered. Path profiles use more history than point profiles. Figure 1 explains these two profiles in detail and Figure 2 shows how to use point and path profiles, respectively, for predicting an end-user's next request.

User sessions

User 1	User 2	User 3
x.html	y.html	x.html
y.html	x.html	z.html
x.html	z.html	y.html
z.html		
y.html		

Session traces

User 1 x->y->x->z->y
User 2 y->x->z
User 3 x->z->y

Point profile

x->y	1
x->z	3
y->x	2
z->y	2

Path profile (for all session traces)

Path	Frequency	Path	Frequency
x->y	1	x->z->y	2
x->y->x	1	z->y	2
x->y->x->z	1	y->x	2
x->y->x->z->y	1	y->x->z	2
x->z	3	y->x->z->y	1

Figure 1: Example of Point and Path profile for User Sessions

User session trace: x->y->?

Selected paths:		**Using point profiles:**		**Using path profiles:**	
x->y->z	9	y->z	9	x->y->z	9
x->y->w	3	y->w	11	x->y->w	3
y->y->w	5	∴ Predict x->y->w		∴ Predict x->y->z	
z->y->w	3				

Figure 2: Example of prediction using point and path profiles

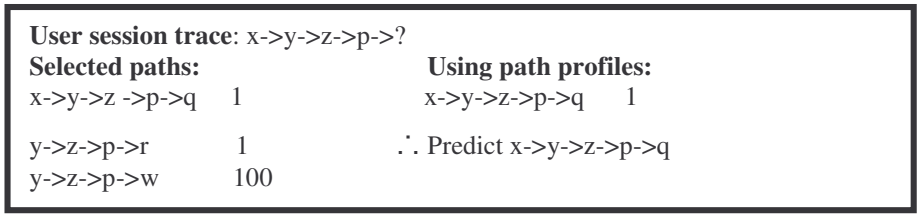

Figure 3: Example of incorrect prediction using path profiles

As seen above, the use of path profiles works better for prediction than do point profiles. It is a reasonable result when considering the fact that path profiles use more history than point profiles. However, path profiles also have some flaws. Figure 3 shows the case when the path profiles do not work well. Because path profiles consider first the matched length first and then frequency, in the above case *q* will be selected. However, *w* is a better prediction. This algorithm can possibly be improved if we use a threshold for filtering the candidate paths.

5. SUGGESTED NEW APPROACH

One of the target issues involved in the increasing of computer performance involves boosting degrees of instruction-level parallelism. However, the biggest performance obstacle to achieving this parallelism is the presence of conditional branch instructions that determine which instructions need to be executed next. This branch latency problem is more severe for modern billion-transistor one-chip microprocessor architectures. Many valuable resources are idling while they await the results of a branch instruction. Therefore, many computer architects have suggested various ways of using the history of previous branch results to predict the path of conditional branches.

Due to the extensive research history of computer architecture, the average branch prediction accuracy for known schemes reaches 94.4 %, and it is insisted that some current algorithms achieve an even greater accuracy of 97 % [15]. In contrast, a current sophisticated Web path prediction based on the Markov algorithm show only 62.5% accuracy [4]. If we can efficiently apply these branch prediction algorithms to an end-user's path prediction without increasing the complexity of the system, there is a great possibility of decreasing a vast amount of end-user perceived delay on the Web and enhancing the correctness of recommendations in knowledge management systems.

5.1. Similarity between the branch prediction and path prediction

The situation of a Web server waiting idly for the user's next request is similar to that of a microprocessor forced to wait for the result of a branch instruction before deciding which section of code should be executed next. All modern high-performance microprocessors adopt branch prediction schemes to speculate on the execution of future instructions. If the prediction is correct, then execution continues, and the latency of the branch execution is totally hidden. Although a branch may be incorrectly predicted, there is little penalty; the machine efficiently discards the results of speculative instructions and starts again, this time executing the correct path instructions.

If a Web server can predict the next URL in advance before a user explicitly requests it, in the same way that the microprocessor predicts the next instruction after branch instruction, then the server could quickly generate the following requested pages. This would hide the latency of requests if the predicted pages are accurate anticipations of a user's future page visits.

5.2. Various branch prediction algorithms

Some branch prediction schemes are static in that they use profiling statistics to make predictions, while others are dynamic in that they use run-time execution history [15].

The most well-known basic technique involves making a prediction based on the direction of the branch executed the last few times. This technique is referred to as *bimodal branch prediction* [16]. Bimodal prediction uses a two-bit, up/down saturating counter. For each branch taken, the counter is incremented; for each non-taken branch, it is decremented. However, because it is a two-bit saturating counter, it is neither incremented over three nor decremented below zero. If the first bit of the counter is one, then it will predict as taken; if zero, it will predict as non-taken. When each branch is biased to one direction, such bimodal prediction works well.

Recent research has discovered the possibility of achieving more accurate prediction by utilizing more branch history. One improved algorithm recognizes that many branches execute repetitive patterns, and it uses these patterns. Because the history used is local to the current branch, this method is called *local branch prediction* [16]. For example, in the following code:

for (i = 0; i <=3; i++) { }

the corresponding branch will show the pattern $(1110)^n$, where 1 represents taken, 0 represents non-taken, and n represents the number of times that the loop is executed. Clearly, if we know the results of three previous branches, then we can always predict the next branch direction. Local prediction works well for simple repetitive patterns.

Another algorithm is called *global branch prediction* [15]. In local branch prediction, only patterns of the current branch are considered. Therefore, Yeh and Patt [15] suggest utilizing other recent branch histories to predict more correctly. This approach combines all recent branches to make a prediction and so, the history is global. Global branch prediction works effectively when the direction taken by sequentially executed branches is highly correlated. One example involves the following code:

if (x < 5)…
if (x > 5)…

With global branch prediction, if the first branch's result is known, the second branch can be predicted easily.

Each branch prediction algorithm has its own advantages. A possible natural response would be to consider combining different prediction algorithms in order to achieve better prediction accuracy. McFarling [8] proposes *Combining Branch Predictors* with this approach. He shows the results of combining various predictors. One useful combination involves bimodal and global branch predictors. In this

combination, a bimodal scheme is used to predict usual branch direction, and the global information can be used if it is worthwhile. The benchmark test results indicate that with regard to performance, combining predictors works more effectively that using a single predictor alone.

6. OTHER RELATED ISSUES

6.1. Empirical analysis for end-users' browsing behaviors to test and verify prediction algorithms

In path profiling, the concept of the user session is critical. However, because most HTTP logs have only the Source IP address field and not the unique user id field for each request, it is very difficult to distinguish whether two requests came from the same user. This situation worsens when a proxy server is set up on a client side; many such IP addresses represent the address of the proxy server that multiplexes many different users' requests. Therefore, an empirical study of end-users' Web interaction behavior is required to verify and augment prediction algorithms. For example, by examining the average amount of time a user interacts with the Web, from when they start surfing until they leave, we can heuristically decide the period of the same user's session. That period will be used to cut off a long session log in order to determine whether two requests from the same user were made during a single visit to a site.

Also, if we decide to provide direct manipulated interfaces for prediction, it is important to know what items should appear in the prediction list, how they should be ordered, how long the list should be, and how it should be displayed. Nor do we know how likely it is that people could find their page on that list. In their empirical study, Tauscher and Greenberg focus on data collection methodology, results, discussions, and implications [14].

6.2. Parameter field in the HTTP log

Currently, many sites provide CGI scripts to communicate in both directions between server and client. These scripts add the parameter field in the user's URL request string, which is separated from the requested file name by the "?" character. These parameter fields maintain much valuable user information, such as address, name, and telephone number. One interesting direction of future work in this field could involve the building of agents to retrieve personal information automatically: they would simply determine which parameters to consider and which to ignore. In addition to predicting paths, these agents could be utilized for personalized recommendation tools.

6.3. Cache replacement algorithms for predicted pages

Client caches prevent downloading the same page twice. However, because the size of a local cache on the client side is limited, it can fill up quickly when an end-user requests more pages. This means that some pages in the cache need to be replaced by newly requested pages. If we replace pages in a cache inefficiently, network traffic will be increased, and an end-user's perception of latency will increase. For example, let us assume that when a cache is full, it replaced the last fetched page. In that algorithm, if an end-user alternates visits to two pages on the Web, the local cache must download the requested page repeatedly because the previously visited page has been deleted from the cache. Therefore, under this cache replacement policy, an end-user will suffer longer periods of latency.

6.4. An efficient storing method for path profiles

The explosive growth of required space for path profiles is no trivial problem. Schechter, et al. [12] point out that storing the profile for a single path of length S (i.e., all the necessary subpaths) requires storage that grows as $O(S^3)$. Due to resource constraints, storing all path profiles end-users have visited is almost impossible. Even though we may have a huge amount of storage, enough to store entire path profiles, searching and retrieving the right path profiles from this information so as to predict the user's next link will continue to be a problem. Therefore, in order to produce an accurate predictive model that uses the least amount of possible space, smart storing methods for path profiles are vital. Some researchers suggest using a compact data structure that tries to address worst-case space constraints [12]. Others suggest removing low information elements from the model [10]. Another possible solution is using a hash table for saving the storage space and reducing the retrieval time.

7. DISCUSSION

It has become a fact of everyday life that the Internet connects people to each other more quickly and efficiently than has ever been possible before. Despite its advances, though, one seemingly insurmountable problem has marred human interaction with the Internet: the long latency period involved in end-users' retrieving from a server a page of interest. This latency often forces end-users to wait idly in front of a monitor and, in the worst case scenario, can make them so frustrated with their ability to interact with the Internet that they end up abandoning the computer. However, while an end-user is viewing a page, we waste the local CPU clock cycles, which could be used more profitably to satisfy end-users' future requests in advance. More specifically, intelligence could be added to servers and Web browsers, so that whenever an end-user chooses a new page, those that are potentially soon to be demanded—ones decided by a prediction algorithm—could be transferred in advance to the client cache. In effect, these documents would be waiting in the wings.

Prediction of Web documents can also be used to organize up-to-date digital repositories for special interest groups. Such groups of Web users who have the same interests constitute what is called a networked community. Intelligent algorithms seem particularly applicable to this pool of end-users. For example, a predicting agent may download all documents related to discussion topics, update them regularly, and make them readily available to all members of the networked community.

Admittedly, it is difficult, although not impossible, to anticipate end-users' behavior while they are surfing the Web. While no software can predict the future exactly, prediction could be performed more effectively, however, if we limit the target subject to a specialized group of end-users, members of the aforementioned networked community. Because a networked community is usually composed of people who share the same specific interests, the way that one member interacts with the Internet would display patterns similar to those generated when other members visit sites. So, analyzing a server's session logs and then prefetching predicted pages could provide efficient solutions for reducing the latency most end-users experience when using the web.

8. REFERENCES

[1] A. Bestavros, "Using speculation to reduce server load and service time on the WWW", **Proceeding of the 4th ACM International**

Conference on Information and Knowledge Management, 1995, pp. 403-410.

[2] S. Brin, & L. Page, "The anatomy of a large-scale hypertextual web search engine", **WWW7 / Computer Networks 30**, 1998, pp. 107-117.

[3] E. Chi, J. Pitkow, J. Mackinlay, P. Pirolli, R. Gossweiler, and S.K. Card, "Visualizing the evolution of web ecologies", **Proceedings of the Conference on Human Factors in Computing systems**, 1998, pp. 400-407.

[4] D. Duchamp, "Prefetching hyperlinks", **Proceedings of USITS'99: the 2nd USENIX Symposium on Internet Technologies & Systems**, 1999, pp. 127-138.

[5] E.P. Markatos, & C.E. Chronaki, "A Top-10 approach to prefetching on the web", **In Proceedings of INET'98**, 1998.

[6] J. Helms, D.C. Neale, P.L. Isenhour, & J.M. Caroll, "Data logging: higher-level capturing and multi-level abstraction of user activities", **Proceedings of 40th Annual Meeting of the Human Factors and Ergonomics Society**, 2000.

[7] D.M. Hilbert, & D.F. Redmiles, D.F. "An approach to large-scale collection of application usage data over the internet", **Proceedings of the 20th International Conference on Software Engineering**, 1998, pp. 136-145.

[8] S. McFarling, "Combining branch predictors", **Western Research Laboratory Technical Note, 36**, 1993.

[9] B. Mobasher, R. Cooley, & J. Srivastava, J. "Automatic personalization based on web usage mining", **Communications of the ACM, 43(8)**, 2000, pp. 142-151.

[10] J. Pitkow & P. Pirolli, Peter. "Mining longest repeating subsequences to predict World Wide Web surfing", **Proceedings of USITS'99: the 2nd USENIX Symposium on Internet Technologies & Systems**. 1999.

[11] R. Sarukkai, "Link prediction and path analysis using Markov chains", **Computer Networks 33(1-6)**, 2000, pp. 377-386.

[12] S. Schechter, M. Krishhan, & M.D. Smith, "Using path profiles to predict HTTP requests", **Proceedings of the Seventh International World Wide Web Conference, 30(1-7)**, 1998, pp. 457-467.

[13] A.C. Siochi, & D. Hix, "A study of computer-supported user interface evaluation using maximal repeating pattern analysis", **Proceedings of Association of Computing Engineering chi'91 Conference on Human Factors in Computing Systems**, 1991, pp. 301-305.

[14] L. Tauscher, & S. Greenberg, "How people re-visit web pages: empirical findings and implications for the design of history systems", **International Journal of Human-Computer Studies, 47(1)**, 1997, pp. 97-138.

[15] T.Y. Yeh, & Y.N. Patt, "Alternative implementations of two-level adaptive branch prediction", **The 19th Annual International Symposium on Computer Architecture**, 1992, pp. 124-134.

[16] J. E. Smith, "A study of branch prediction strategies", **In Proc. of 8th International Symposium in Computer Architecture**, 1981, pp. 135-148.