

## Chapter

### A semi-random prediction scenario for user requests\*

Ernst-Georg Haffner, Institute of Telematics, Trier, Germany, haffner@ti.fhg.de

Uwe Roth, Institute of Telematics, Trier, Germany, roth@ti.fhg.de

Thomas Engel, Institute of Telematics, Trier, Germany, engel@ti.fhg.de

Christoph Meinel, Institute of Telematics, Trier, Germany, meinel@ti.fhg.de

#### Abstract

*Since the introduction of the World Wide Web (WWW) at the beginning of the 90s, the Internet community has grown steadily and is now more than 1000 times larger as it was in the beginning. To meet the growing requirements for an easy and efficient information retrieval from the Internet, technologies that allow prediction of user requests are gaining more and more importance. Many approaches exist to predict static HTML pages with Markov Chains, but more general models are needed to depict pure data (such as dynamic HTML) requests adequately. In this paper we present a scenario to model, test and classify user requests that is based upon a controlled strategy, also referred to as semi-random strategy.*

**Keywords:** Prediction, Internet performance, WWW, request classification

#### 1 Introduction

The World Wide Web (WWW) spreads very fast. Beside static HTML pages, dynamically processed data gain more and more importance as well. Such information is mostly retrieved through the Common-Gateway-Interface (CGI) [8]. On the server side, scripts are executed, usually processing database requests. To enable browsers accessing the results of such queries, the web server has to translate them into HTML [13]. Lately, applications that achieve communication directly via specially designed data servers - preferably using Java Applets ([7],[15]) have been introduced to the net. Making use of this method, translation of data into HTML is no longer required - in fact, it is not even desired!

Such servers boast a number of sophisticated technologies to meet the demands arising from complex information sources assembled on the Internet. A concept called *Prediction* used for decades in compiler construction and VLDB-Design [11], is now being discussed for Internet use. The basic goal of using prediction algorithms is to reduce the delay the use experiences when waiting for a reply from a server.

There are several known approaches for HTML requests containing hyperlinks from one document to another. Scientific literature describes manifold ways to determine the probability of HTML-page requests - for instance *Path Profiling* [14]. Making use of a *Markov-Model*, the tree-like connections can be approximately modeled and is an adequate approach [10]. Beside the modeling of stochastic processes as discrete Markov-Chains ([1] and [2]), a continuous chain approach is also being examined [11]. In the general case - with no hyperlinks between data requests - we have to work with general forms of *Conditional Probabilities*.

Using prediction in compiler construction has almost no negative side effects. For Internet use, this is no longer true though. A wrong use of prediction algorithms may lead to a deterioration of network and system load. It is necessary to operate *defensively* and to take safety precautions in order to reduce negative effects

---

\* In Proc. "Asia Pacific International Web Conference", APWeb99, Hong Kong, China, 1999, pp. 11-18

of incorrect predictions ([4] and [5]).

It is, of course, a good idea to test the quality of a prediction algorithm with *real life* server request logs. However, to classify the different kinds of user behavior from *random* (not predictable) to *semi-random* (predictable), we will present a test scenario as a model of the client-server web communication. The basic idea is to manage both the random and the constant part of user requests. Depending upon the data of a web server and the average computable user classification, the prediction algorithms should be able to adapt their cost functions and thresholds - or should even be switched off if user behavior is not predictable!

In the next section we will elaborate on a general scenario of prediction requests. The *randomness and semi-randomness model* is presented in section 3. An analysis of the results will follow in section 4. A summary and an outlook conclude the paper.

## 2 A general prediction scenario

### 2.1 Cost function and threshold

To calculate the profit of prediction we will focus only on the aspect of *Precalculation* (of time intensive operations). Our results are easily transferable to *Prefetching* (of data from server to client) and *Piggyback Transmission* (of meta data together with requested data; for details see [6] and [3]). The following method for calculating the cost function and the threshold corresponds to many known approaches ([12], [9]).

We will speak of the term *session s* as a sequence of user requests within a fixed time span  $\mathbf{T}$ . A customary value would be 30 minutes (e.g. [14]).

The function  $\gamma_s$  investigating the cost of a session  $\mathbf{s}$  in the case of precalculation can be seen as:

$$\gamma_s(s) = \sum_{i=1}^A (\gamma_R(d_i) + \gamma_Z(d_i)) \cdot p_i \quad (1)$$

where  $\mathbf{A}$  calculations of data sets  $d_1, d_2, \dots, d_A$  occur with a probability of  $p_i$  within  $\mathbf{s}$ .

The costs on the server side can be divided into *avoidable* (or *optimisable*) costs  $\gamma_Z$  and inherent *resource* costs  $\gamma_R$ . For instance, sorting of data takes a certain amount of time, thus producing costs ( $\gamma_R$ ). Deteriorating effects for the user arise when system load is high. If a server could manage to sort data when system load is low, the corresponding avoidable costs ( $\gamma_Z$ ) would depend on the moment of computation. Thus only  $\gamma_Z$  can be reduced by means of prediction.

Making use of prediction the cost function  $\gamma_s$  looks as follows:

$$\gamma_s(s) = \sum_{i=1}^B \gamma_R(d_i) + \sum_{i=1}^A \gamma_P(d_i) + \sum_{i=B+1}^A (\gamma_R(d_i) + \gamma_Z(d_i)) \cdot p_i \quad (2)$$

where  $B \leq A$  and without restriction of universality the first  $B$  data sets  $d_1, d_2, \dots, d_B$  could be predicted. The share of  $\gamma_R$  is increased as a result of prediction calculations which have to be made, regardless of whether they will be needed later on or not. The avoidable costs of  $d_{B+1}, \dots, d_A$ , however, can then be ignored ("correctly predicted"). On the other side a new cost value  $\gamma_P$  which describes the costs of the prediction ("bookkeeping costs") has to be taken into account.

Prediction only makes sense if (2) is smaller than (1) for any data set  $d_i$ ,  $i \in \{1, \dots, B\}$ . Thus, the threshold for the probability value can be calculated as:

$$p_i > \frac{\gamma_{\mathcal{R}}(d_i) + \gamma_{\mathcal{P}}(d_i)}{\gamma_{\mathcal{R}}(d_i) + \gamma_{\mathcal{Z}}(d_i)} = \frac{1 + \frac{\gamma_{\mathcal{P}}(d_i)}{\gamma_{\mathcal{R}}(d_i)}}{1 + \frac{\gamma_{\mathcal{Z}}(d_i)}{\gamma_{\mathcal{R}}(d_i)}} = \frac{1 + S_o(d_i)}{1 + S_f(d_i)} \quad (3)$$

$S_o$  describes the *Prediction-Overhead* that has to be small in comparison to  $S_f$  (*System Flexibility*) for a low threshold.

## 2.2 Prediction algorithm to determine the request probability

The requests of a user session  $s$  is modeled - without considering the order or repeating requests - as a binary  $\mathbf{A}$ -vector of data set indices:

$S = (s_i)$  with:  $s_i = 1$  if the data set  $d_i$  was requested by the client;  $s_i := 1$  otherwise

With other words: On server side, there exist  $\mathbf{A}$  different types of possible data sets and the element of a session vector is "1" (set) if and only if the corresponding data set has been requested one or more times within time span  $T$ .

In our scenario the prediction store consists of a symmetric, quadratic matrix  $\varphi$ , where the elements are frequencies of requests. The matrix  $\varphi$  is also called *Memory Matrix*.

$\varphi = (\varphi_{ij})$  with  $1 \leq i, j \leq \mathbf{A}$ .  $\varphi$  is initialized as 0-matrix:  $\varphi_0 = (\varphi_{ij}) = 0 \quad \forall 1 \leq i, j \leq \mathbf{A}$

Within every session  $s$  the matrix  $\varphi$  is calculated to  $\varphi'$ , so that the elements of  $\varphi$  represent the frequencies of data requests, depending on requests from other data sets (rows and columns of the matrix).

$$\varphi' = (\varphi'_{ij}) = \begin{cases} \varphi_{ij} + 1 & \text{iff } s_i = 1 \wedge s_j = 1 \\ \varphi_{ij} & \text{otherwise} \end{cases}$$

The conditional probability  $p_i$ , stating that  $d_i$  will be requested, if  $d_j$  has been requested during the same session  $s$  can be calculated as:

$$p_i(d_i | d_j) = \frac{\varphi'_{ij}}{\varphi_{ij}} \quad (4)$$

However, we do not focus on the efficiency of that algorithm in this stage but use it as a (general) basis to model prediction in a test scenario. An interesting aspect is the modeling of user vectors in such a scenario.

## 3 Modeling randomness and semi-randomness

### 3.1 Introduction to the idea of semi-randomness

Requests of data can be divided into two classes: *predictable requests* and *random requests*. The reason for that classification lies in the different *a priori request probability*. Predictable requests are often based on measurable personal facts of certain importance for a user so that the request probability is not purely random, but *semi-random*.

Let us consider the case of financial data - especially of stock values - as a first example. It is obvious that the customer of a bank wants to be informed about the values of shares that are part of her/his personal portfolio. Perhaps all stock data values are of a *certain interest*, but those of the personal portfolio are of a *special interest*. For security reasons it might not be possible to access portfolio data but when the proposed method is used, accessing the portfolio is not even necessary! It is sufficient to take into account that there is *potential* semi-random data to provide. For another example we should have a look at great sports events, for instance the Football World Cup. The preliminary rounds are mostly more interesting for people that have the same nationality as the football players on the playground. There are also examples with no objective semi-randomness. Weather forecasts can be of extremely high interest for someone who plans to renew the roof of her/his house but there are no measurable facts that can be used by algorithms to predict the high interest in such data.

Of course the distinction into random and semi-random requests is not very sharp. There can be practical cases where no real difference between both types of a priori probability exist. It is still important to model random and semi-random requests in a different way.

### 3.2 A classifying prediction scenario

To test prediction algorithms, we build a test scenario where the session vectors are generated with a *randomizer-module*. Without restriction, we can say that the first  $m$  elements of the session  $A$ -vectors ( $d_1, \dots, d_m$ ) are considered as *predictable*, the others ( $d_{m+1}, \dots, d_A$ ) as *casual*.  $S = (d_1, \dots, d_m, d_{m+1}, \dots, d_A)$

Two global parameters control the generation of binary values: the *random factor*  $R$  and the *density*  $D$  with:

$$R \in [0,1] \text{ and } 1 \leq D \leq A$$

The random factor  $R$  controls the probabilities of all values of the session vector  $S$ . " $R = 1$ " means "pure randomness" with a constant probability of all elements of  $S$ . " $R = 0$ " means "no randomness" where the first  $m$  elements of  $S$  are always 1, all other values are 0.

The density  $D$  describes the average number of values set to 1 within the session vectors with the highest randomness ( $R = 1$ ).  $D$  plays no role if  $R$  equals 0. Thus the importance of  $D$  increases with the value of  $R$ .

In general, the probabilities of the elements of the generated session vectors follow the condition below:

$$1 \leq i \leq m : P(d_i = 1) = 1 + R \cdot (D / A - 1) \text{ and } m+1 \leq i \leq A : P(d_i = 1) = R \cdot (D/A) \quad (5)$$

$$\text{For a random factor of } 1 (R = 1) \text{ we get the probabilities: } 1 \leq i \leq A : P(d_i = 1) = D/A \quad (6)$$

whereas a random factor of  $R = 0$  leads to:

$$1 \leq i \leq m : P(d_i = 1) = 1 \text{ and } m+1 \leq i \leq A : P(d_i = 1) = 0 \quad (7)$$

## 4 Presentation of instructive test results

### 4.1 Test volume and training phase

Our results are based on thousands of test-runs with different settings of the global parameters  $A$ ,  $m$ ,  $D$ ,  $R$ , and of the number of initial training sessions  $T$  terminating with  $P$  prediction sessions. Each run generated a session vector  $S$ . Elements of  $S$  that are set to 1 represent a data request. In our scenario the order of those requests is not modeled. Thus every " $d_i = 1$ " of  $S$  could potentially be the first request of  $S$  as the base for a prediction of the rest. For that reason we generated several prediction calculations, one for each "1" in  $S$  without using information about the settings of the other elements.

The prediction algorithm as presented in section 2 produces a probability for every element of  $S$ . Any calculated value exceeding the system dependent threshold  $p_i$  has to be considered as "predicted". In our test

scenario we used the (constant) threshold  $\delta := 0.95$  and  $\delta := 0.85$  as reasonable settings of  $p_i$ . The complete prediction vector has been compared afterwards with the "real" session vector. After P test-runs we evaluated the following values:

- OP:** Overall sum of predictions (of whole sessions)
- CP:** sum of correctly predicted settings  
("1" in the prediction vector corresponded to "1" in the session vector)
- WP:** sum of wrongly predicted settings  
("1" in the prediction vector corresponded to "0" in the session vector)
- CN:** sum of correctly prediction of non-settings  
("0" in the prediction vector corresponded to "0" in the session vector)
- WN:** sum of wrongly prediction of non-settings  
("0" in the prediction vector corresponded to "1" in the session vector)

High values of WP must be viewed as critical because they can cause enormous negative side effects whereas high values of WN are not critical. However, they are a signal for inefficient prediction mechanisms.

The first results during the test runs were unexpected. We had been thinking that it would be a good idea to settle a training phase at the beginning of the tests to fill the Memory Matrix with reasonable values. Astonishingly, the predictive correctness in the training phase had no significant effect on the results of the predicting phase. The main reason for this is that prediction will not occur if the corresponding line of the Memory Matrix is 0. This is always the case at the beginning of the training phase.

#### 4.2 The effect of random factor and density

The following tables show some test results with different global parameter settings. We begin with variable values of m and two different settings of  $\delta$  (table 1). The ratio  $PQ = CP/WP$  is called the *prediction quality*.

A	m	D	R	P	$\delta$	PQ	OP	CP	WP	CN	WN
$10^3$	$10^0$	$10^2$	0.1	$10^3$	<b>0.95</b>	0.5	9917	5505	10386	$9.8 \cdot 10^6$	101956
$10^3$	$10^1$	$10^2$	0.1	$10^3$	<b>0.95</b>	3.6	18045	52986	14896	$1.7 \cdot 10^7$	282609
$10^3$	$10^2$	$10^2$	0.1	$10^3$	<b>0.95</b>	8.9	99084	657088	74200	$8.9 \cdot 10^7$	9178672
$10^3$	$10^0$	$10^2$	0.1	$10^3$	<b>0.85</b>	0.6	9921	6591	10499	$9.7 \cdot 10^6$	100870
$10^3$	$10^1$	$10^2$	0.1	$10^3$	<b>0.85</b>	5.9	18066	136187	22934	$1.8 \cdot 10^7$	199408
$10^3$	$10^2$	$10^2$	0.1	$10^3$	<b>0.85</b>	10.1	98997	$8.6 \cdot 10^6$	858014	$8.8 \cdot 10^7$	1195560

Table 1: Prediction results with different share of predictable values

We can see here that the prediction quality PQ increases with the share of predictable elements m. The threshold value of  $\delta = 0.95$  seems too pessimistic, especially for high values of m.

The importance of the random factor R is presented below (table 2).

A	m	D	R	P	$\delta$	PQ	OP	CP	WP	CN	WN
$10^2$	$10^1$	$10^1$	<b>0.0</b>	$10^5$	0.95	$\infty$	99990	899910	0	$9.0 \cdot 10^6$	0
$10^2$	$10^1$	$10^1$	<b>0.1</b>	$10^5$	0.95	10.3	99942	22170	2146	$8.9 \cdot 10^6$	894584
$10^2$	$10^1$	$10^1$	<b>0.2</b>	$10^5$	0.95	4.7	100081	5174	1108	$8.9 \cdot 10^6$	929835
$10^2$	$10^1$	$10^1$	<b>0.3</b>	$10^5$	0.95	2.5	99912	2498	1003	$8.9 \cdot 10^6$	942867
$10^2$	$10^1$	$10^1$	<b>0.4</b>	$10^5$	0.95	1.3	100058	1246	977	$8.9 \cdot 10^6$	958507
$10^2$	$10^1$	$10^1$	<b>0.5</b>	$10^5$	0.95	0.7	100039	658	894	$8.9 \cdot 10^6$	968370
$10^2$	$10^1$	$10^1$	<b>0.6</b>	$10^5$	0.95	0.4	99891	339	911	$8.9 \cdot 10^6$	972688

Table 2: Influence of the random factor upon prediction quality

It is obvious that low values of R lead to very good prediction conditions, but even a random factor of 0.4 is not acceptable because there are almost as many wrongly predicted requests (WP) as correctly predicted ones (CP).

The density D is variable in the next table (3), again with different settings of the threshold  $\delta$ .

A	m	D	R	P	$\delta$	PQ	OP	CP	WP	CN	WN
$10^3$	$10^1$	$10^0$	0.1	$10^3$	0.95	8.7	9003	1429	165	$8.9 \cdot 10^6$	72473
$10^3$	$10^1$	$10^1$	0.1	$10^3$	0.95	5.8	9383	4421	760	$9.3 \cdot 10^6$	81480
$10^3$	$10^1$	$10^2$	0.1	$10^3$	0.95	3.6	18045	52986	14896	$1.8 \cdot 10^7$	282609
$10^3$	$10^1$	$10^3$	0.1	$10^3$	0.95	10.6	107887	$1.1 \cdot 10^6$	101682	$9.6 \cdot 10^7$	$1.1 \cdot 10^7$
$10^3$	$10^1$	$10^0$	0.1	$10^3$	0.85	9.1	8997	71477	7871	$8.9 \cdot 10^6$	2425
$10^3$	$10^1$	$10^1$	0.1	$10^3$	0.85	8.9	9401	74515	8418	$9.3 \cdot 10^6$	11386
$10^3$	$10^1$	$10^2$	0.1	$10^3$	0.85	5.9	18053	136187	22934	$1.8 \cdot 10^7$	199408
$10^3$	$10^1$	$10^3$	0.1	$10^3$	0.85	10.6	106939	$1.1 \cdot 10^6$	101687	$9.6 \cdot 10^7$	$1.1 \cdot 10^7$

Table 3: Importance of the density factor

The density factor D shows a very interesting effect of the prediction quality PQ. For very low values of D, PQ is very good. This becomes clear if we regard D as a kind of unpredictable part of a session vector. But when D reaches very high values, nearly as high as A, the part of "randomly" correctly predicted (CP) requests increases as well. Thus D can be viewed as a key parameter of the whole prediction process.

The main and most evident test results are summarized below.

- There is a strong relationship between the randomness of user behavior and the possibility to predict upcoming events. Prediction itself should be turned off if the amount of semi-random data (m, table 1) is becoming too small due to the negative side effects of wrong prediction.
- Even a moderate value of the random factor R leads to a catastrophic prediction profit (table 2).
- Increasing the amounts of semi-random elements (m) and decreasing the density (D) to a very low value has a desirable effect upon the prediction results.
- As we stated above the number of training sessions is of no significant importance for the correctness of the ensuing prediction. In addition we found that even the amount of session vectors in the prediction phase plays no important role for the quality of the prediction.
- The threshold that is determined by equation (3) controls the prediction results in a decisive manner. But to lower the threshold values does not only lead to a better result of correct hits (CP) but also accounts for the increasing amounts of the very negative wrong prediction (WP).

To evaluate the quality of the prediction values we must have a closer look at the prediction quality (PQ) quotient. All values with  $PQ < 1.0$  are evidently not preferable. We found that in practice,  $PQ > 5.0$  is a good signal to improve user perceived performance with prediction methods.

## 5 Summary and outlook

Prediction of client requests for the Internet seems to be a promising way to reduce user perceived latency. We have described the foundations of the prediction theory and have presented a controlled or semi-random test scenario. There are many possibilities to implement prediction algorithms, but one should never forget that there are not only many chances but also dangerous risks, especially when data is prefetched.

Our idea was to model and classify user behavior with means of a controlled random test scenario. A first analysis of the test-runs with semi-random and random requests has yielded some interesting results. The classification of user behavior may serve as a possibility to decide whether to use prediction algorithms or not. Further researches will hopefully deliver new answers to current questions of prediction theory. The

usefulness of applying prediction algorithms and the decision to stop the guessing of upcoming user requests must happen on the basis of user behavior classification.

## References

- [1] Azer Bestavros, Speculative Data Dissemination and Service, *Proceedings of ICDE96*, March 1996, <http://cs-www.bu.edu/faculty/best/res/papers/>
- [2] Azer Bestavros, Using Speculation to Reduce Server Load and Service Time on the WWW, *Proceedings of CIKM95*, November 1995, <http://cs-www.bu.edu/faculty/best/res/papers/>
- [3] Paul Barford, Mark Crovella, Generating Representative Web Workloads for Network and Server Performance Evaluation, *SIGMETRICS 1998*, <http://www.acm.org/pubs/citations/proceedings/metrics/277851/p151-barford/>
- [4] Mark Crovella, Paul Barford, The Network Effects of Prefetching, *IEEE Infocom 1998*, <http://www.cs.wpi.edu/~webbib/bibentries/infocommisc.html>
- [5] Ramón Cáceres, Fred Douglass, Anja Feldmann, Gideon Glass, Michael Rabinovich, Web Proxy Caching: The devil is in the details, *Workshop on Internet Server Performance*, June 1998, Madison, WI, <http://akpublic.research.att.com/~douglass/papers/>
- [6] Edith Cohen, Balanchander Krishnamurthy, Jennifer Rexford, Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters, *SIGCOMM 1998*, [http://www.acm.org/sigcomm/sigcomm98/tp/abs\\_20.html](http://www.acm.org/sigcomm/sigcomm98/tp/abs_20.html)
- [7] Java, <http://www.javasoft.com/>
- [8] J. Depp, *Developing CGI Applications with Perl*, WILEY-VCH, 1996
- [9] Zhimei Jiang, Leonard Kleinrock, An Adaptive Network Prefetch Scheme, *IEEE J Sel Areas Commun*, 1998, <http://millennium.cs.ucla.edu/LK/Bib/bib.html>
- [10] Achim Kraiss, Gerhard Weikum, Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions, *Proceedings of the 23<sup>rd</sup> VLDB Conference*, 1997, <http://sunsite.informatik.rwth-aachen.de/dblp/db/conf/vldb/vldb97.html#KraissW97>
- [11] Achim Kraiss, Gerhard Weikum, Integrated document caching and prefetching in storage hierarchies based on Markov-chain predictions, *The VLDB Journal*, Springer-Verlag, 1998, <http://link.springer.de/link/service/journals/00778/bibs/8007003/80070141.htm>
- [12] Thomas Kroeger, Darrell Long, Jeffrey Mogul, Exploring the bounds of web latency reduction from caching and prefetching, 22. *USENIX*, December 1997, <http://csl.cse.ucsc.edu/reports/>
- [13] José M. Martínez, Francisco Morán. Catalog: a WWW Gateway for DBMSs, *Web Net 96*, 1996, <http://aace.virginia.edu/aace/conf/webnet/proc96index.html>
- [14] Schechter, Krishnan, Smith, Using path profiles to predict HTTP requests, *Proceedings of the World Wide Web Conference*, 1998, <http://decweb.ethz.ch/WWW7/1917/com1917.htm>
- [15] Prashant Sridharan, *Advanced Java Networking*, Prentice-Hall, 1997