

WEB TRAFFIC MODELING AND ITS APPLICATION IN THE  
DESIGN OF CACHING AND PREFETCHING SYSTEMS

by

Abdullah Balamash



A Dissertation Submitted to the Faculty of the  
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements  
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2 0 0 4

UMI Number: 3145040

## INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

**UMI<sup>®</sup>**

---

UMI Microform 3145040

Copyright 2004 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

The University of Arizona ®  
Graduate College

As members of the Final Examination Committee, we certify that we have read the


dissertation prepared by Abdullah Saeed Balamash

entitled Web Traffic Modeling And Its Application In The

Design Of Caching And Prefetching

and recommend that it be accepted as fulfilling the dissertation requirement for the

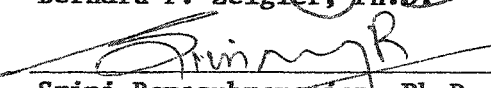
Degree of Doctor Of Philosophy

  
Marwan Krunz, Ph.D.


Aug. 20, 04  
date

  
Bernard P. Zeigler, Ph.D.

Aug 20/04  
date

  
Srini Ramasubramanian, Ph.D.

Aug 20, '04.  
date

  
Bane Vasic, Ph.D.

8/20/04

date

  
Young-Jun Son, Ph.D.

08/20/2004  
date

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

  
Dissertation Director: Marwan Krunz, Ph.D.

Aug. 25, 04  
date

### STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: \_\_\_\_\_

A handwritten signature in dark ink, consisting of several loops and a long horizontal stroke at the end, positioned above the signature line.

## ACKNOWLEDGEMENTS

First of all, I would like to thank my parents and my wife, for without whom this dissertation would not have been possible. Their support, patience, and understanding helped me through the difficult times. Their encouragement and faith gave me the strength to pursue my study and complete the degree requirements.

I would like to express my deepest gratitude to my advisor Dr. Marwan Krunz for his invaluable assistance and support. Special thanks to my friends Mohammad Hassan, Dr. Turgay Korkmaz, Alaa Muqattash, and Aytac Azgin who in some way have contributed to this dissertation.

Finally, I would like to thank my dissertation committee members Dr. Bernard Zeigler, Dr. Srinivasan Ramasubramanian, Dr. Bane Vasic, and Dr. Young Jun Son for spending their valuable times in my PhD exams and making constructive comments that led to the improvement of the contents of this dissertation.

## DEDICATION

*To my parents and my wife*

## TABLE OF CONTENTS

<b>LIST OF FIGURES</b> . . . . .	<b>8</b>
<b>LIST OF TABLES</b> . . . . .	<b>10</b>
<b>ABSTRACT</b> . . . . .	<b>11</b>
<b>CHAPTER 1 INTRODUCTION</b> . . . . .	<b>13</b>
1.1 Motivation . . . . .	13
1.2 Main Contributions . . . . .	15
1.3 Dissertation Overview . . . . .	15
<b>CHAPTER 2 BACKGROUND</b> . . . . .	<b>17</b>
2.1 WWW Traffic Characteristics . . . . .	17
2.1.1 Popularity . . . . .	17
2.1.2 Temporal Locality . . . . .	18
2.1.3 Spatial Locality . . . . .	20
2.1.4 Self-Similarity . . . . .	21
2.1.5 Multifractality . . . . .	23
2.2 WWW Client Behavior . . . . .	25
2.3 Approaches to Reduce WWW Response Time . . . . .	25
<b>CHAPTER 3 MULTIFRACTAL MODELING OF WWW RE-</b> <b>QUESTS</b> . . . . .	<b>31</b>
3.1 Introduction . . . . .	31
3.2 Multifractal Analysis of WWW Traffic . . . . .	33
3.2.1 Riedi et al.'s Multifractal Model . . . . .	34
3.2.2 Reducing the Number of Parameters . . . . .	38
3.2.3 Selecting the Random Variable $A_j$ . . . . .	43
3.3 Stack distance Model . . . . .	44
3.3.1 Extracting the Empirical Normalized Stack Distance String . . . . .	44
3.3.2 Modeling the Normalized Stack Distance String . . . . .	46
3.3.3 Modeling Popularity and Generating Synthetic Reference Strings . . . . .	50
3.3.4 Performance Evaluation . . . . .	53
3.4 Inter-request Distance Model . . . . .	61
3.4.1 Traffic Generation Process . . . . .	63
3.4.2 Performance Evaluation . . . . .	64

## TABLE OF CONTENTS – *Continued*

<b>CHAPTER 4 WWW PREFETCHING MODEL . . . . .</b>	<b>69</b>
4.1 Introduction . . . . .	69
4.2 Prefetching System Architecture . . . . .	69
4.3 Analysis of Prefetching Gain . . . . .	72
4.4 Prefetching Gain Optimization . . . . .	75
4.4.1 Prefetching Precision Independent of $N_p$ . . . . .	77
4.4.2 Prefetching Precision Varies with $N_p$ . . . . .	78
4.5 Prefetching Protocol . . . . .	81
4.5.1 Traffic Prediction . . . . .	83
4.6 Effect of Caching on Prefetching Gain . . . . .	84
4.7 Simulation Results . . . . .	88
4.7.1 Simulation Setup . . . . .	88
4.7.2 Traffic Model . . . . .	88
4.7.3 Prediction Model . . . . .	90
4.7.4 Validation of $\Delta_h$ and $\rho_p$ . . . . .	91
4.7.5 Validating the Access Improvement Index . . . . .	92
<b>CHAPTER 5 CONCLUSIONS AND FUTURE WORK . . . . .</b>	<b>99</b>
5.1 Conclusions . . . . .	99
5.2 Future Work . . . . .	100
<b>APPENDIX A PROOF OF THEOREM 3.2.1 . . . . .</b>	<b>102</b>
<b>APPENDIX B PROOF OF THEOREM 4.4.1 . . . . .</b>	<b>103</b>
<b>APPENDIX C PROOF OF THEOREM 4.4.2 . . . . .</b>	<b>106</b>
<b>APPENDIX D PROOF OF THEOREM 4.4.4 . . . . .</b>	<b>108</b>
<b>REFERENCES . . . . .</b>	<b>109</b>



## LIST OF FIGURES

2.1	Frequency of file accesses versus file ranking. . . . .	18
2.2	Impact of transforming the marginal distribution of a F-ARIMA model on the correlation structure. . . . .	22
2.3	Self-similar traffic versus Poisson traffic. . . . .	24
2.4	Possible locations for deploying WWW caching. . . . .	26
2.5	Markovian predictor. . . . .	30
3.1	Accuracy of the multifractal model in capturing the ACF of the stack distance string of a real WWW trace. . . . .	32
3.2	Process of generating the scaling coefficients in the DWT. . . . .	36
3.3	Fitting the correlation structure of the normalized stack distance string (Calgary trace). . . . .	40
3.4	Fitting the correlation structure of the normalized stack distance string (ClarkNet trace). . . . .	41
3.5	Fitting the correlation structure of the normalized stack distance string (Worldcup98 trace). . . . .	42
3.6	Example showing our approach for extracting stack distances from a real trace. . . . .	46
3.7	Marginal distribution of the normalized stack distance string (ClarkNet trace). . . . .	47
3.8	Marginal distribution of the normalized stack distance string (Calgary trace). . . . .	48
3.9	Marginal distribution of the normalized stack distance string (Worldcup98 trace). . . . .	49
3.10	Algorithm for generating synthetic scaled stack distance strings. . . . .	51
3.11	Algorithm for generating synthetic WWW strings. . . . .	53
3.12	File miss ratio versus cache size (Clarknet trace). . . . .	55
3.13	Byte miss ratio versus cache size (Clarknet trace). . . . .	56
3.14	File miss ratio versus cache size (Calgary trace). . . . .	57
3.15	Byte miss ratio versus cache size (Calgary trace). . . . .	58
3.16	File miss ratio versus cache size (Worldcup98 trace). . . . .	59
3.17	Byte miss ratio versus cache size (Worldcup98 trace). . . . .	60
3.18	WWW synthetic trace generation process. . . . .	65
4.1	Components of the prefetching system. . . . .	71
4.2	Client behavior. . . . .	72

# LIST OF FIGURES – *Continued*

4.3	$I$ versus $N_p$ ( $r = 500$ kbps, $C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $P_{th} = 0.5$ , $h_c = 0$ , $h_{proxy} = 0$ ). . . . .	78
4.4	$I$ versus $N_p$ ( $C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $h_c = 0$ ). . . . .	79
4.5	$I$ versus $N_p$ for the case $\bar{P} = \frac{4.5(1-e^{-0.24N_p})}{N_p}$ ( $C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $P_{th} = 0.5$ , $h_c = 0$ ). . . . .	80
4.6	Effects of $R$ and $h_{proxy}$ on $I$ for the case $\bar{P} = \frac{4.5(1-e^{-0.18N_p})}{N_p}$ ( $C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 9$ files, $P_{th}(R = 1, h_{proxy} = 0) = 0.5$ , $h_c = 0$ ). . . . .	81
4.7	Impact of proxy caching on the effectiveness of prefetching ( $C = 1000$ kbps, $\lambda = 20$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $h_c = 0$ ). . . . .	85
4.8	Impact of local (regular) caching on the effectiveness of prefetching ( $r = 1000$ kbps, $C = 1000$ kbps, $\lambda = 20$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $h_{proxy} = 0$ ). . . . .	86
4.9	Impact of local (regular) caching on the effectiveness of prefetching ( $r = 56$ kbps, $C = 1000$ kbps, $\lambda = 20$ files/sec, $\bar{s} = 40$ Kbits, $N_{on} = 15$ files, $h_{proxy} = 0$ ). . . . .	87
4.10	Client-side traffic generation process. . . . .	90
4.11	Mechanism for traffic prediction. . . . .	91
4.12	Increase in the client's cache hit ratio due to prefetching versus $N_p$ when $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ( $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$ , $h_c = 0.31$ ). . . . .	92
4.13	Average system load versus $N_p$ for the case $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ( $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$ , $h_c = 0.31$ ). . . . .	93
4.14	$I$ versus $N_p$ ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ , $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$ , $h_c = 0.31$ ). . . . .	95
4.15	$I$ versus $N_p$ under fixed-size files ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ , $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$ , $h_c = 0.31$ ). . . . .	96
4.16	$I$ versus $N_p$ ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ , $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$ , $h_c = 0.31$ ). . . . .	97
4.17	$I$ versus $N_p$ ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ , $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$ , $h_c = 0.31$ ). . . . .	98
B.1	Approximation of $f_R(\rho)$ by $1/(1 - \rho^R)$ . . . . .	104

## LIST OF TABLES

3.1	Upper bound on the ratio $E[(X^{(m)})^2]/E[(X^{(2m)})^2]$ for various distributions. . . . .	44
3.2	Summary of the data sets used in the modeling study. . . . .	54
3.3	Statistical comparisons for CLARKNET trace. . . . .	61
3.4	Statistical comparisons for CALGARY trace. . . . .	62
3.5	Statistical comparisons for WORLDCUP98 trace. . . . .	62
3.6	Cache miss ratio inaccuracy for the worldcup98 trace. . . . .	66
3.7	Cache miss ratio inaccuracy for ClarkNet trace. . . . .	67
3.8	Statistical comparisons for the worldcup98 trace. . . . .	67
3.9	Statistical comparisons for ClarkNet trace. . . . .	68
4.1	Parameters of the prefetching protocol. . . . .	83
4.2	Probability distributions used in the simulations. . . . .	89

## ABSTRACT

Network congestion remains one of the main barriers to the continuing success of the Internet. For web users, congestion manifests itself in unacceptably long response times. One possible remedy to the latency problem is to use caching at the client, at the proxy server, or even within the Internet. However, documents on the World Wide Web (WWW) are becoming increasingly dynamic (i.e., have short lifetimes), which limits the potential benefit of caching. The performance of a WWW caching system can be dramatically increased by integrating document prefetching (a.k.a., “proactive caching”) into its design.

Prefetching reduces the perceived user response time, but it also increases the network load, which in turn may increase the response time. One main goal of this dissertation is to investigate this tradeoff through a mathematical model of a WWW caching/prefetching system, and to demonstrate how such a model can be used in building a real prefetching system. In our model, the client cache consists of a “regular” cache for on-demand requests and a “prefetching cache” for prefetched requests. A pool of clients connect to a proxy server through bandwidth-limited dedicated lines (e.g., dialup phone lines). The proxy server implements its own caching system. Forecasting of future documents is performed at the client based on the client’s access profile and using hints from servers. Our analysis sheds light on the interesting tradeoff between aggressive and conservative prefetching, and can be used to optimize the parameters of a combined caching/prefetching system. We validate our model through simulation. From the analysis and/or simulation, we find that: (1) prefetching *all* documents whose access probabilities exceed a given threshold value may, surprisingly, degrade the delay performance, (2) the variability of WWW file sizes has a detrimental impact on the effectiveness of prefetching, and (3) coexistence between caching and prefetching is, in general, beneficial for the

overall performance of the system, especially under heavy load.

Ideally, a caching/prefetching system should account for the intrinsic characteristics of WWW traffic, which include temporal locality, spatial locality, and popularity. A second contribution of this dissertation is in constructing a stochastic model that accurately captures these three characteristics. Such a model can be used to generate synthetic WWW traces and assess WWW caching/prefetching designs. To capture temporal and spatial localities, we use a modified version of Riedi et al.'s multifractal model, where we reduce the complexity of the original model from  $O(N)$  to  $O(1)$ ;  $N$  being the length of the synthetic trace. Our model has the attractiveness of being parsimonious (characterized by few parameters) and that it avoids the need to apply a transformation to a self-similar model (as often done in previously proposed models), thus retaining the temporal locality of the fitted traffic. Furthermore, because of the scale-dependent nature of multifractal processes, the proposed model is more flexible than monofractal (self-similar) models in describing irregularities in the traffic at various time scales.

## CHAPTER 1

### INTRODUCTION

#### 1.1 Motivation

WWW users can experience response times in the order of several seconds. Such response times can be quite unacceptable, causing some users to request the delayed files over and over again. This, in turn, aggravates the situation and further increases the network load and the perceived latency. Caching is an effective approach for reducing the user-perceived response time by, storing copies of popular WWW files in a local cache, a proxy server cache close to the end user, or even within the Internet. However, the benefit of caching diminishes as WWW files become more dynamic [31, 52, 35]. A cached file may become stale at the time of its request, because most WWW caching systems in use today are *passive* (i.e., files are fetched or validated only when requested). Moreover, users mostly click on hyperlinks to other files. If a hyperlinked file is accessed for the first time by the client and has not been requested by other clients who have a common cache with this client, then the file must be retrieved from the original server. The main consequence of these limitations is that the maximum hit ratio that can be achieved by any caching system is limited and was found not to exceed 50% [1].

Prefetching (or proactive caching) aims at overcoming the limitations of passive caching by proactively fetching files in anticipation of users future requests. In addition to regular (static) files, the prefetched files may also include hyperlinked files that have not been requested before as well as dynamic objects <sup>1</sup>[79, 68]. Moreover, since WWW files are frequently updated, stale cached files can be considered for prefetching, as well.

---

<sup>1</sup>A dynamic object is a web file that is created by the server on the fly based on a certain number of parameters provided by the user

Although prefetching can increase the overall cache hit ratio, resulting in reduced user response time, it consumes additional network resources because some prefetched files are never requested. This, in turn, can increase the response time if prefetching is not performed carefully. The prefetching problem has two aspects. The first one is how to specify the set of files that are most likely to be accessed in the near future. The second aspect is how to determine the number files to prefetch that would minimize the overall average response time.

Most of the studies on WWW prefetching dealt with the first part of the problem where a set of candidate files for prefetching is computed based on a prediction algorithm (see section 2.3). Each candidate file is identified by its name and its likelihood of being accessed in the near future. The likelihood value of a file represents the probability of accessing such file in the near future. In these studies, either a threshold-based approach is used, whereby all files with access probabilities that exceed a fixed threshold value are prefetched, or a fixed number of the most popular files are prefetched. Both approaches may actually *increase* the average response time, since they do not consider the state of the network during the prefetching process [88, 24]. One solution to this problem is to *dynamically adjust the prefetching threshold* and to vary the number of files to prefetch depending on the network state. The need for such a solution is one motivation behind this dissertation.

An good WWW caching/prefetching system must take into account in its design the intrinsic properties of WWW traffic. These properties include temporal locality, spatial locality, and popularity. Temporal locality measures the closeness in time between requests to the same file. Spatial locality measures the correlation between requests to different files (e.g., if file *A* is currently being requested, then there is a good chance that file *B* will be requested in the near future). Popularity refers to the overall likelihood of requesting a particular file, independent of other files. See section 2.1 for detailed descriptions of these properties.

The ability to assess the performance of WWW caching/prefetching system hinges on the availability of a representative workload that can be used in trace-driven simulations [18, 46]. Measured (“real”) traces can also be used for this

purpose. However, due to the difficulty associated with capturing real traces, only a handful of such traces are available in the public domain (see [102] for public domain traces). This makes it hard to provide simulation results with reasonable statistical credibility. A more feasible alternative is to rely on synthetic traces that are derived from an approximate stochastic model. The need for such a model is another motivation behind this dissertation.

## 1.2 Main Contributions

The main contributions of this dissertations include the following:

- Investigation of the suitability of using multifractal modeling techniques in capturing the essential properties of WWW traffic.
- Development of two parsimonious (characterized by few parameters) multifractal traffic models that accurately capture the essential WWW traffic properties.
- Mathematical modeling and analysis of a generic, hybrid prefetching/caching system and investigation of the tradeoff between the increased network load and the reduced user response time using such system.
- Investigation of the impact of caching on the effectiveness of prefetching.
- Development of an adaptive prefetching protocol that can optimize the client access latency based on the network state.

## 1.3 Dissertation Overview

In chapter 2, we provide a background of related topics that help discuss the remaining chapters of this dissertation. This background includes the main WWW traffic properties and overview of WWW caching and prefetching mechanisms.

We investigate the suitability of multifractal modeling in capturing the WWW essential traffic properties in Chapter 3. We develop two stochastic WWW traffic



models that can be used for performance evaluation of caching/prefetching systems. Our models exploits the versatility of multifractal processes to simultaneously capture the intrinsic properties of WWW traffic, namely temporal locality, spatial locality, and popularity. They have the attractiveness of being parsimonious (characterized by few parameters) and they avoid the need to apply nonlinear transformation to the ‘basic’ process, as often done in previous models [13], thus retaining the temporal locality of the fitted traffic. Because of the scale-dependent nature of multifractal processes, the proposed models are more flexible than a monofractal (self-similar) models in describing irregularities in the traffic at various time scales.

In Chapter 4, we investigate the effectiveness of client-side prefetching in the presence of local and proxy caching systems. We analytically study the tradeoff between the increased network load and the reduction in the user response time due to prefetching. We then use our analysis to design a prefetching protocol that can optimize the user response time by dynamically adjusting the system parameters based on the network state.

We conclude this dissertation in Chapter 5 and provide some directions for future research.

## CHAPTER 2

### BACKGROUND

#### 2.1 WWW Traffic Characteristics

In our context, WWW traffic represents the stream of clients requests seen by a WWW/proxy server. Understanding the properties of such a stream is key to designing good caching/prefetching systems [4]. In the following subsections, we describe these characteristics in details.

##### 2.1.1 Popularity

Popularity refers to the overall likelihood of requesting a particular file, independent of other files. It was shown in [7, 17] that popular files on the WWW are “very popular” meaning that references to a few number of files represent a high percentage of the total number of requests in a given stream of WWW requests. Several other studies showed that the distribution of the number of references a WWW file gets follows a Zipf-like distribution [41, 27, 4, 12]. Zipf’s law was applied earlier in characterizing the frequency of using a given word in terms of its popularity rank [93]. It states that the relationship between a word’s popularity rank ( $\rho$ ) and the frequency ( $P$ ) of using this word in a given text is given by:

$$P \sim \frac{1}{\rho^\alpha}$$

where  $\alpha = 1$ . For a Zipf-like distribution, the constant  $\alpha$  was found to take values between 0 and 1 [17]. Thus, the  $n$ th most popular file is  $2^\alpha$  times as likely to be accessed as the  $(2n)$ th most popular file. Figure 2.1 demonstrates the application of Zipf-like distribution to WWW requests. The figure shows the number of times that a file has been accessed versus the rank of the file in a given WWW trace, where rank 1 means the most frequently accessed file. Note that both axes are in the log

scale. It is clear that the curve fits a straight line reasonably well. The straight line on the log-log scale indicates that the number of requests is proportional to  $1/\rho^\alpha$  where  $-\alpha$  is the slope of the line. For this plot, the value of  $\alpha = 0.65$  was obtained using the mean-square error curve-fitting technique.

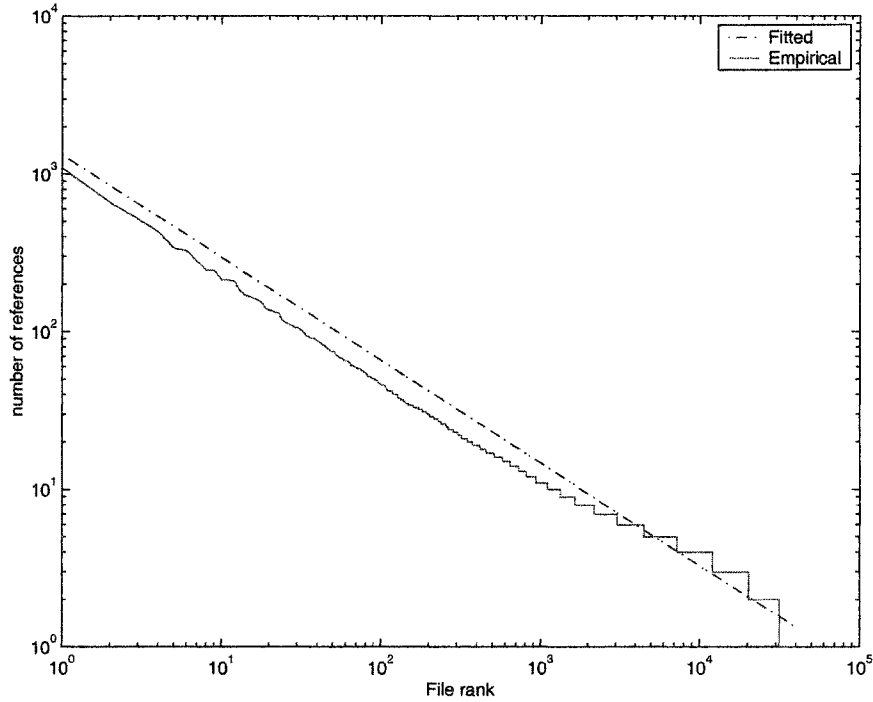


Figure 2.1: Frequency of file accesses versus file ranking.

### 2.1.2 Temporal Locality

Temporal locality refers to the likelihood that a file that has just been requested will be requested again in the near future. In other words, it measures the closeness in time between requests to the same WWW file. Accurate characterization of temporal locality is very important because utilizing such a property in the design of cache replacement policies can significantly improve the performance of a caching system.

WWW traffic characterization has been the focus of several previous studies;

examples of which are given in [7, 4, 13, 27, 48, 47]. In these studies, the temporal locality of the traffic was represented by the marginal distribution of the stack distance string [58]. The stack distance string, which is an equivalent representation of a reference string, is obtained by transforming the reference string using the least recently used (LRU) stack, as follows. Let the reference string be  $R_t = \{r_1, r_2, \dots, r_t\}$ , where  $r_j$  is the file (or object) requested at time  $j$ . Note that a file may appear multiple times in  $R_t$ . Let the LRU stack at time  $t$  be  $S_t = \{Obj_1, Obj_2, Obj_3, \dots, Obj_n\}$ , where  $Obj_1, Obj_2, \dots, Obj_n$  are distinct files;  $Obj_1$  is the most recently requested file,  $Obj_2$  is the second most recently requested file, and so on. Let  $d_t$  be the stack distance of the file referenced at time  $t$  (the position of the file in the LRU stack at time  $t - 1$ ). Whenever a reference is made to a file, the LRU stack must be updated. If  $r_{t+1} = Obj_i$ , then the LRU stack becomes  $S_{t+1} = \{Obj_i, Obj_1, Obj_2, \dots, Obj_{i-1}, Obj_{i+1}, \dots, Obj_n\}$  and  $d_{t+1} = i$ . Thus, for any reference string  $R_t = \{r_1, r_2, \dots, r_t\}$ , there is a corresponding stack distance string  $D_t = \{d_1, d_2, \dots, d_t\}$ .

In [49], the authors studied the temporal locality of WWW traffic and concluded that such a phenomenon is induced by both temporal correlations and long-term popularity. More specifically, references to *globally* popular files tend to be close to each other in time. Furthermore, references to certain *unpopular* files may exhibit strong temporal correlations, if these references appear “clustered” in time (e.g., a file may be frequently requested but only during a short period of time). Long-term popularity suggests the use of the number of requests each file gets in caching decision (e.g., the Least Frequently Used caching policy (LFU)). On the other hand, the existence of temporal correlation suggests the use of the time since last access in caching decision (e.g., the Least Recently Used caching policy (LRU)). Accordingly, It is important to differentiate between the two aspects of temporal locality since this can help in cache design [48, 49]. The standard stack-distance-string approach does not differentiate between the two sources of temporal locality, since the distribution of the stack distance string is predominantly affected by the popularity profile (i.e., long-term popularity). To address this problem, one needs

to neutralize the effect of popularity. The authors in [19] introduced a new measure for temporal locality, called the *scaled stack distance*, that attempts to do that. The scaled stack distance string is obtained by normalizing the stack distances by their expected values (assuming that requests to a given file are evenly distributed over the duration of the trace). Accordingly, the scaled stack distance string captures the deviation of the stack distances from their expected values, which is a measure of the clustering of the references. In other words, equally popular files would have the same expected stack distance but would differ in their scaled stack distances depending on the degree of short-term correlations. It was found that the distribution of both, the normalized and the non-normalized stack distance follows a lognormal-like distribution. The following simplified example explains the concept. The effect of long-term popularity is seen in the reference stream *ABACAADAEAAAEAF*, in which temporal locality arises from the popularity of file *A*. The effect of short-term temporal locality can be seen in the reference stream *AABBCCDDEEFF*, in which temporal locality arises from the repetition pattern of the files.

### 2.1.3 Spatial Locality

Spatial locality measures the correlation between requests to different files (e.g., if file *A* is currently being requested, then there is a good chance that file *B* will be requested in the near future). The reference stream *ABC..ABE..ACB..ADEB..* illustrates this concept where with high probability file *B* is requested after file *A* being requested. Accurate characterization of spatial locality is fundamental to the prediction of future requests.

In [4] the authors showed that spatial locality can be captured (at least, in part) through the autocorrelation structure (ACF) of the stack distance string. They argued that the stack distance string exhibits a *long range dependent* (LRD) behavior. An LRD behavior is implied by a persistent correlation where the summation of the ACF over all lags is infinite. To *simultaneously* model the marginal distribution (temporal locality) and the correlation structure (spatial locality) of the stack-distance string, they relied on the work in [42], which proved the invariance

of the Hurst parameter (a parameter that describes the asymptotic behavior of the ACF) to transformations of the marginal distribution of an LRD process. More specifically, the authors in [42] proved that under some mild assumptions, a point-by-point transformation  $Y = F_y^{-1}(F_x(X))$  of a *Gaussian* self-similar process (see section 2.1.4 for details on self-similar processes)  $X$  with Hurst parameter  $H$  results in a self-similar process  $Y$  with the same Hurst parameter, where  $F_x$  and  $F_y$  are the CDFs for  $X$  and  $Y$ , respectively. It should be noted, however, that this result is valid asymptotically and only for Gaussian processes (e.g., fractional ARIMA). More importantly, while this result assures the invariance of  $H$ , it does not necessarily preserve the shape of the ACF. As an example, consider the transformation of the Gaussian distribution of a F-ARIMA model into a lognormal distribution (this distribution adequately models the marginal distribution of the stack distance string [48, 19]). The resulting ACFs are shown in Figure 2.2, along with the ACF of the WWW stack distance string of the CLARKNET trace (see section 3.3.4 for details on the trace information). The figure illustrates two main drawbacks of the transformation-based approach. First, while the transformation may capture the asymptotic behavior of the ACF (the  $H$  parameter), it destroys the overall shape of the original ACF of the F-ARIMA model. Second, the original F-ARIMA model itself is not accurate in representing the real ACF at finite lags.

To avoid these problems, in this dissertation, we resort to multifractal modelling to simultaneously capture the correlation structure and the marginal distribution of the stack distance string.

#### 2.1.4 Self-Similarity

Self-similarity is a traffic property that is manifested by the presence of burstiness at all time scales, compared to the non-self-similar traffic (e.g., the Poisson traffic), which smooths off at large time scales as can be seen in Figure 2.3. In the figure, we show a self-similar time series and a poisson time series at different aggregation levels (time scales), 1, 16, 128, and 1024. As can be seen that the self-similar traffic is still bursty even at time scale 1024, while the poisson traffic almost smoothed out.

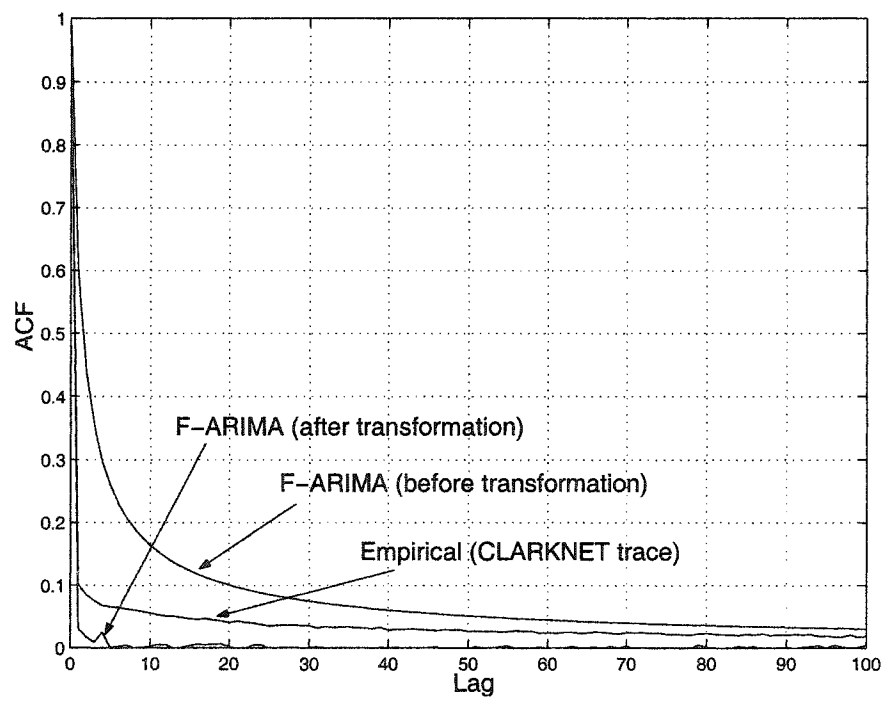


Figure 2.2: Impact of transforming the marginal distribution of a F-ARIMA model on the correlation structure.

A self-similar time series has the property that when aggregated the new time series asymptotically has the same ACF as the original. In other words, given a stationary time series  $X = (X_t; t = 0, 1, 2 \dots)$ , we define  $m$ -aggregated series by summing the original series  $X$  over non-overlapping blocks of size  $m$  as follows

$$X_k^{(m)} = \frac{1}{m} \sum_{i=(k-1)m+1}^{km} X_i, \quad k = 1, 2, \dots \quad (2.1)$$

Then if  $X$  is self-similar, it asymptotically has the same ACF as  $X^{(m)}$ . This means that the distribution of the aggregated series is the same as that of the original except for changes in scale ( $X \stackrel{d}{=} m^{1-H} X^{(m)}$ ,  $0.0 < H < 1.0$ ).

An attractive feature of a self-similar process is that the degree of self-similarity is expressed by a single parameter, the Hurst parameter ( $H$ ). This parameter expresses the rate of decay of the autocorrelation function, which is the same for all aggregated versions of the traffic time series.

An LRD process is a self-similar process that has a slowly (hyperbolically) decaying ACF ( $H > 0.5$ ). In an LRD time series, the summation of the autocorrelation values over all lags approaches infinity. Moreover, the variance of  $n$  samples does not decrease as a function of  $n^{-1}$ , which is the case for uncorrelated data, but rather as function of  $n^{-\beta}$ ,  $0 < \beta < 1$ .

The works in [53, 69, 90] pointed out that a self-similar traffic is constructed by multiplexing a large number of ON/OFF sources that have heavy-tailed ON and OFF periods. For WWW traffic in particular, Crovella and Bestavros [25, 26] showed an evidence that the traffic exhibits self-similar behavior. The existence of self-similarity was attributed to the ON/OFF behavior of WWW clients and the heavy-tailed distributions of WWW file sizes, the ON periods, and the OFF periods.

### 2.1.5 Multifractality

Multifractality is a generalization of self-similarity (monofractality), whereby the Hurst parameter is not fixed, but varies with time scale. This variability makes multifractal processes more flexible than monofractal processes in describing “ir-



regularities” in the traffic. Several studies demonstrated that wide-area network traffic exhibits a multifractal scaling behavior [85, 56, 75, 36, 39, 35, 33, 97, 64]. Feldmann et al. [36] explained the multifractal behavior of network traffic through *multiplicative processes* or *cascades*. Cascading is a process by which a component (number, mass) is fragmented into smaller and smaller components according to some rule, and at the same time fragments the mass associated with these components according to other rule. Random cascades were introduced by Mandelbrot as a model for turbulence [55]. The hierarchical structure of today’s networks with their protocols can be viewed as a cascade process that fragments units of data at one layer into smaller units at the next layer [36]. The limiting construct of the fragmentation process is a multifractal structure. Comprehensive discussions of multifractal processes can be found in [37, 76, 40, 74] and the references therein.

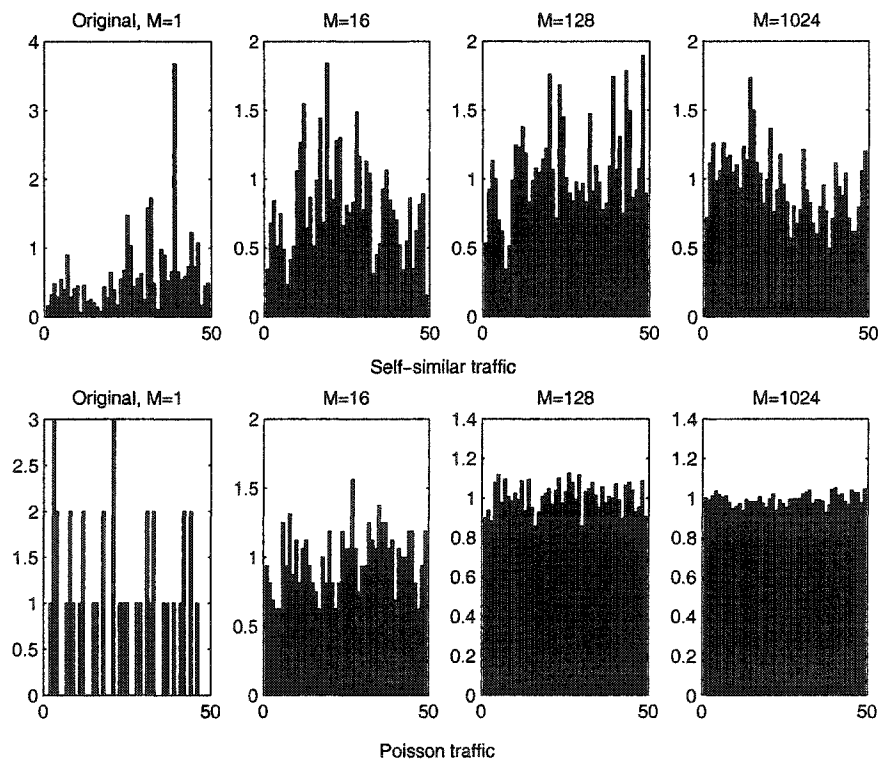


Figure 2.3: Self-similar traffic versus Poisson traffic.

## 2.2 WWW Client Model

The authors in [13] described a WWW client as an ON/OFF process where it alternates between ON and OFF periods. The ON period is the period during which files are being transferred, and the OFF period is the period the user spends reading a retrieved document (an html file and its inline objects) before clicking on another WWW document. The OFF period was found to conform to a heavy-tailed distribution [12, 27, 60].

A random variable  $X$  is said to have a heavy-tailed distribution,  $F$  if

$$\lim_{x \rightarrow \infty} (1 - F(x)) \sim x^{-\alpha}, \quad 0 < \alpha < 2. \quad (2.2)$$

Significance of heavy-tailed distribution is that they result in bursty user traffic [90]. The length of an ON period depends on the number of embedded files in an html document, the sizes of these files, and the way a client is connected to the Internet. The number of embedded files in a WWW document plus the main WWW file was found to follow a heavy-tailed distribution [13, 60].

Knowing the sizes of WWW files is useful in the design of caching systems. Although there is a difference between sizes of files hosted by WWW servers and the sizes of file transfers from servers to clients, both characteristics were found to be fairly close [6, 12]. Previous works have shown that file sizes follow a heavy-tailed distribution [61, 12, 27, 60].

## 2.3 Approaches to Reduce WWW Response Time

WWW caching provides a good solution for the WWW latency problem by bringing documents closer to clients. As mentioned before, caching can be deployed at various points in the Internet: within the client browser, at or near the server (reverse proxy) to reduce the server load, or at a proxy server. A proxy server is a computer that is often placed near a gateway to the Internet (see Figure 2.4) and that provides a shared cache to a set of clients. Client requests arrive at the proxy regardless of the WWW servers that host the required documents. The proxy either serves these

requests using previously cached responses or obtains the required documents from the original Web servers on behalf of the clients. It optionally stores the responses in its cache for future use. Hence, the goal of proxy caching is twofold. First, proxy caching reduces the access latency for a document. Second, it reduces the amount of external traffic that is transported over the wide-area network (primarily from servers to clients), which also reduces the user's perceived latency. A proxy cache may have limited storage in which it stores popular files. Whenever the cache is full and the proxy needs to cache a new file, it has to decide which file to evict from the cache to accommodate the new file. The policy used for the eviction decision is referred to as the *replacement policy*.

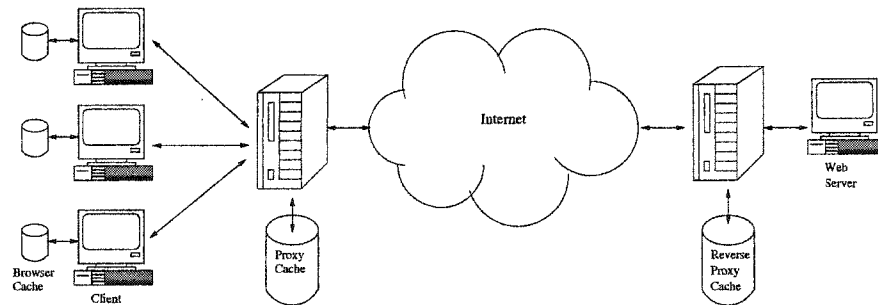


Figure 2.4: Possible locations for deploying WWW caching.

Caching in the web works in a similar manner to that of a traditional memory system [95]. However, caching policies for traditional memory systems do not necessarily perform well when applied to WWW traffic for the following reasons:

- In memory systems, caches deal mostly with fixed-size pages, so the size of the page does not play any role in the replacement policy. In contrast, WWW files vary in size, and the file size can affect the performance of the policy [92, 44].
- The cost of retrieving missed WWW files from their original servers depends on several factors, including the distance between the proxy and the original servers, the size of the file, and the bandwidth between the proxy and the original servers. Such dependence does not exist in traditional memory systems.

- WWW files are frequently updated, which means that it is very important to consider the file expiration date at replacement instances. In memory systems, pages are not generally associated with expiration dates.
- As discussed before, the popularity of WWW files follows a Zipf-like law. Accordingly, file popularity needs to be considered in any WWW caching policy to optimize a desired performance metric. A Zipf-like law has not been noticed in memory systems

Several WWW replacement policies have been proposed in the literature to deal with the above issues (see for example, [1, 89, 91, 18, 8, 63, 38, 5, 6, 2, 81, 78, 80, 49, 46, 82, 72, 9, 71, 11]).

While WWW caching is considered to be passive (i.e., files are fetched or validated only when requested), Prefetching refers to proactive fetching of WWW files in anticipation of users future requests. Prefetching can be deployed in three ways:

1. Between client and WWW server [15, 28, 24, 3, 22, 66, 50, 65, 83].
2. Between WWW proxy and WWW server [52, 43, 20, 57, 21].
3. Between client and WWW proxy [54, 34].

Regardless of where prefetching is deployed, the key metric used to evaluate a given prefetching scheme is the latency seen by the end user when requesting a file.

A prefetching scheme can be “non-speculative”, whereby a list of specific files are prefetched. For example, the user can specify a list of files to be downloaded at the beginning of any new browsing session. Examples of such mechanisms can be seen in commercial products such as PeakJet2000 [99], NetAccelerator [103], NetSonic [100], Webcelerator [96], and CacheFlow [98]. Non-speculative mechanisms can consume extra network resources because some prefetched files are never requested. Moreover, the number of prefetched files does not depend on the network state, which may worsen the performance for highly loaded systems. Speculative prefetching is a way to reduce the amount of wasted resources due to prefetching files that are never used

[16, 65, 59, 52, 54, 24, 57, 34, 94, 79, 32, 87]. In these mechanisms, a prediction algorithm is used to restrict prefetching to a list of candidate files that are most likely to be requested in the near future. The prediction algorithm compiles a list of candidate files and the probabilities of accessing these files in the near future.

Most speculative prefetching approaches are based on a fixed threshold where only files with a likelihood of being accessed exceeds certain static threshold value are prefetched. These approaches may actually *increase* the average response time, since they do not consider the state of the network when prefetching [88, 24].

There are few prefetching protocols that consider the negative effect of prefetching. Davison et al. [30] proposed a prefetching scheme that uses a connectionless protocol. They assumed that prefetched data are carried by low-priority datagrams that are treated differently at intermediate routers. Although such prioritization is possible in both IPv6 and IPv4, it is not widely deployed. Kokku et al. [51] proposed the use of the *TCP-Nice* congestion control protocol [86] for low-priority transfers to reduce network interference. They used an end-to-end monitor to measure the spare capacity of the server. The reported results show that careful prefetching is beneficial, but the scheme seems to be conservative because it uses an additive increase(increase by 1), multiplicative decrease conservative policy to decide about the amount of prefetching. Crovella et. al [24] showed that a rate-control strategy for prefetching can help reduce traffic burstness and queuing delays. Our work is different from these works since it adopts a dynamic approach that tries to optimize the effect of prefetching without being so conservative. Moreover, it is based on a thorough analysis of the system parameters that can control the prefetching gain.

The authors in [45, 84] consider a dynamic threshold-based prefetching solution, assuming a group of clients who share a single access to the Internet and within only a single level of caching (browser cache). Furthermore, they implicitly assumed that clients have high-bandwidth connections relative to the capacity of the shared access link . This assumption led to the conclusion that it is enough to specify a threshold value for prefetching based on the network condition and then prefetch *all* documents with access probabilities that exceed that value.

As discussed above, speculative prefetching approaches rely on good prediction algorithms to achieve their goals. Such algorithms are based on information about the past. The sources of this information are the client, the proxy server, and the WWW server. Most prediction algorithms track the patterns of accesses by using some variations of *Markov modeling*. A Markovian algorithm represents the files accessed by a client as a string of their IDs. The recurrence of a part of a certain pattern triggers the prefetching of the remainder of the pattern. Such algorithms are implemented using a weighted directed graph where each node represents a file (see Figure 2.5). The weights of the edges usually represent probability values. For example, the weight 0.3 of the edge from node  $A$  to node  $C$  says that if file  $A$  is requested, then with probability 0.3 file  $C$  will be requested next or in the near future. Such algorithms can only predict the next file or some files that are likely to be accessed in the near future. Examples of these algorithms are found in [16, 65, 62]. Markovian algorithms discussed above are considered first-order algorithms in the sense that they only consider the client's last access in predicting the next request. On the other hand, there are some algorithms that consider the last  $m$  requests in their prediction [79, 94, 70]. One such algorithm called Prediction-by-Partial-Matching (PPM) [14, 29, 67, 34, 66] is based on the Partial Matching data compressor. In contrast to Markovian algorithms, there are other algorithms that make use of the structure information of the WWW documents (i.e., hyperlinks that connect different files) [45, 32].

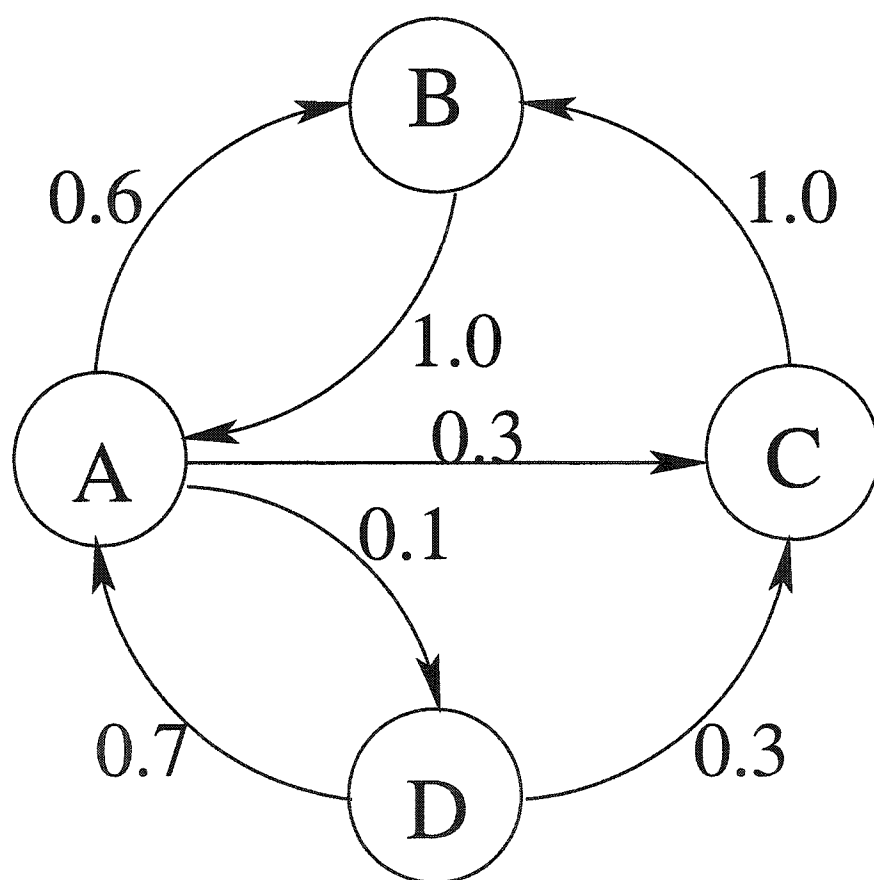


Figure 2.5: Markovian predictor.

## CHAPTER 3

### MULTIFRACTAL MODELING OF WWW REQUESTS

#### 3.1 Introduction

A representative WWW workload helps to judge the performance of WWW caching/prefetching protocols using trace-driven simulations [18, 17, 46, 49]. Moreover, such workload can help in the process of WWW servers design and resource dimensioning. Real traces can be used for this purpose but they lack the flexibility of testing “what if” scenarios that need to be considered for future planning purposes. A more feasible alternative is to rely on stochastic models that can generate representative synthetic workloads. The need for such models is the main motivation behind this modeling work. In this chapter, by WWW traffic we mean the sequence of WWW files served by a WWW server in response to clients’ requests. The popularity profile (how frequently each file is requested) is assumed to be given, as it can be computed easily from the WWW server log files or it can be modeled using a zipf’s-like law.

In this chapter, we present a modified version of Riedi et al.’s multifractal model [74]. We use this modified version to simultaneously capture the temporal and spatial localities of WWW traffic. Riedi’s model has the attractiveness of being able to simultaneously approximate the (lognormal) marginal distribution and the correlation structure of the traffic. Its main disadvantage is its complexity, which grows linearly with the size of the generated trace. We modify this model, reducing its complexity to  $O(1)$ . The resulting model is parsimonious in that it is characterized by four to five parameters representing the mean, variance, and correlation structure of the “normalized stack distance” string. Figure 3.1 shows the accuracy of the multifractal model (which we describe in Section 3.2) in capturing the ACF of the stack-distance string of a WWW trace. The popularity profile of the traffic



is incorporated in the model during the trace generation phase, assuming that the popularity profiles for all files are given beforehand. Our model is mainly intended for offline generation of the traffic demand *seen by a WWW server*. Accordingly, the popularity profiles can be easily computed from the server logs.

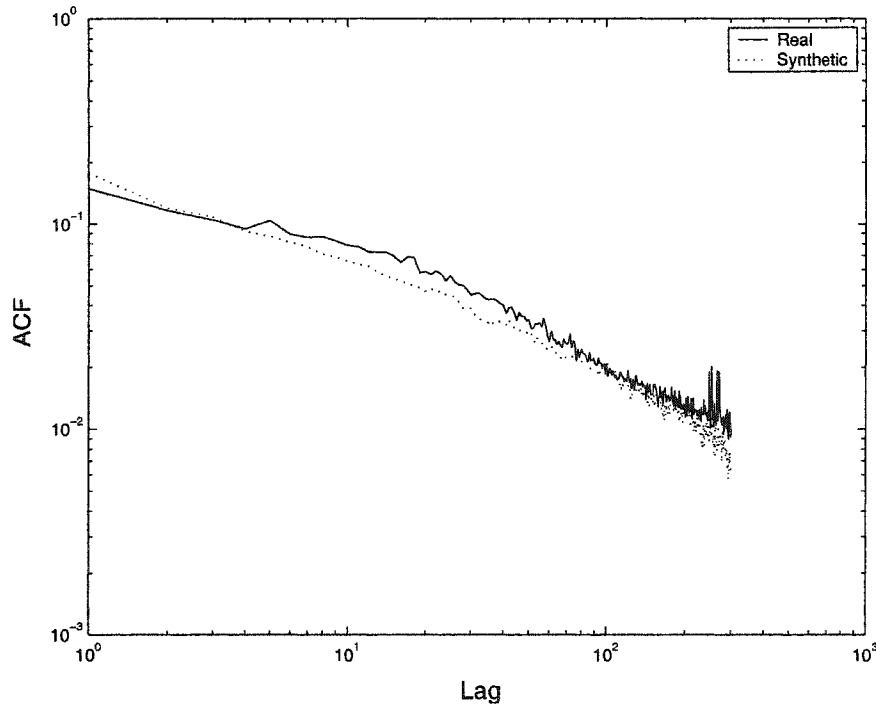


Figure 3.1: Accuracy of the multifractal model in capturing the ACF of the stack distance string of a real WWW trace.

In [74] the authors used a wavelet-based construction of a multifractal process to show that the correlation behavior of a strongly correlated time series can be approximately captured by appropriately setting the second moments of the wavelet coefficients of the multifractal process. This result provides the basis for modeling the ACF of the stack distance string. Combined with the fact that the above multifractal model exhibits an approximately lognormal marginal distribution, it can be used to model both the temporal and spatial localities in WWW traffic.

As was described in Chapter 2, the normalized stack distance string has a lognormal-like distribution and a slowly decaying correlation structure (i.e., LRD

behavior). We employ the multifractal model to capture both the marginal distribution and the correlation structure of the normalized stack distance string. We use extensive simulations to evaluate the performance induced by our WWW traffic model and contrast it with the self-similar model in [4] and the model in [19], using the original (real) traces as a point of reference. Our evaluation measures include sample statistics of the synthetic traces (e.g., mean, variance, correlations, percentiles) as well as the cache and byte hit ratios for a trace-driven LRU cache. The results indicate marked improvement in accuracy when using the proposed multifractal-based WWW model.

The transformation used to generate the stack distance string has an effect on the accuracy of the model. We demonstrate this effect considering an alternative to the stack distance string, namely the inter-request string. File inter-request string is defined as the number of requested files between any two references to the same file.

The rest of the chapter is organized as follows. In Section 3.2 we give a brief overview of Riedi et al.’s multifractal model and the modification we make to it to render it parsimonious. The proposed WWW traffic generation model, which considers the stack distance string approach is given in Section 3.3 with simulation results. The alternative proposed model, which considers the inter-request string approach is described in Section 3.4 along with simulation results that contrast the model to its stack distance version.

### 3.2 Multifractal Analysis of WWW Traffic

As indicated earlier, multifractality is a generalization of monofractality (self-similarity), where the fixed (scale independent)  $H$  parameter of a self-similar process is now scale dependent. The variability in the  $H$  value gives added flexibility to multifractal processes, allowing them to characterize irregularities in the data being modeled. Furthermore, certain multifractal processes, including the one considered in this work, inherently exhibit an approximately lognormal-like marginal distribu-

tion, in line with the shape of the (fitting) marginal distribution of typical WWW traces. This convenient feature allows us to avoid the risky step of transforming the marginal distribution, leaving us with the task of fitting the ACF. In this section, we first briefly describe Riedi et al.'s multifractal model [74]. This model uses a wavelet-based construction to approximately capture the correlation behavior of a given time series by appropriately setting the second moments of the wavelet coefficients at each scale. Its main deficiency is its complexity, which grows linearly (in the number of parameters) with the size of the generated trace. We then describe how we modify this model to reduce its complexity to  $O(1)$ , and then we apply the modified model to characterize the temporal and spatial localities of WWW traffic.

### 3.2.1 Riedi et al.'s Multifractal Model

Riedi et al.'s model relies heavily on the discrete wavelet transform. The idea behind the wavelet transform is to express a signal (time function)  $X(t)$  by an approximated (smoothed) version and a detail. The approximation process is repeated at various levels (scales) by expressing the approximated signal at a given level  $j$  by a coarser approximation at level  $j - 1$  and a detail. At each scale, the approximation is performed through a scaling function  $\phi(t)$ , while the detail is obtained through a wavelet function  $\psi(t)$ . More formally, a wavelet expansion of the signal  $X(t)$  is given by:

$$X(t) = \sum_k U_{J,k} \phi_{J,k}(t) + \sum_{j=J}^{\infty} \sum_k W_{j,k} \psi_{j,k}(t) \quad (3.1)$$

where

$$W_{j,k} \triangleq \int_{-\infty}^{\infty} X(t) \psi_{j,k}(t) dt \quad (3.2)$$

$$U_{j,k} \triangleq \int_{-\infty}^{\infty} X(t) \phi_{j,k}(t) dt \quad (3.3)$$

and  $\psi_{j,k}$  and  $\phi_{j,k}$ ,  $j, k = 0, 1, 2, \dots$ , are *shifted* and *translated* versions of the wavelet and scaling functions  $\psi(t)$  and  $\phi(t)$ , respectively, and are given by:

$$\psi_{j,k}(t) \triangleq 2^{-j/2} \psi(2^{-j}t - k) \quad (3.4)$$

$$\phi_{j,k}(t) \triangleq 2^{-j/2} \phi(2^{-j}t - k). \quad (3.5)$$

In (3.1), the index  $J$  indicates the coarsest scale (the lowest in detail). The coefficients  $W_{j,k}$  and  $U_{j,k}$  are called, respectively, the wavelet and scale coefficients at scale  $j$  and time  $2^j k$ . Together, they define the discrete wavelet transform of the signal  $X(t)$ , assuming that  $\phi(t)$  and  $\psi(t)$  are specified. Several wavelet and scale functions have been used in the literature, giving rise to different wavelet transforms. One popular (and simple) transform is the Haar wavelet transform. This transform, which is specified by the coefficients  $W_{j,k}$  and  $U_{j,k}$  for all  $j$  and  $k$ , can be obtained recursively as follows (we adopt the same convention of [74], where the higher the value of  $j$ , the better is the approximation of the original signal):

$$U_{j,k} = \frac{U_{j+1,2k} + U_{j+1,2k+1}}{\sqrt{2}} \quad (3.6)$$

$$W_{j,k} = \frac{U_{j+1,2k} - U_{j+1,2k+1}}{\sqrt{2}} \quad (3.7)$$

To initialize the recursion, the values of  $U_{j,k}$ ,  $k = 0, 1, \dots, 2^j - 1$ , at the highest value of  $j$  are taken as the empirical trace to be modeled. Figure 3.2 depicts the generation process of the scale coefficients (from top to bottom).

In order to generate synthetic traces with a given autocorrelation structure, the Haar transform is reversed by rewriting (3.6) and (3.7) as:

$$U_{j+1,2k} = \frac{U_{j,k} + W_{j,k}}{\sqrt{2}} \quad (3.8)$$

$$U_{j+1,2k+1} = \frac{U_{j,k} - W_{j,k}}{\sqrt{2}} \quad (3.9)$$

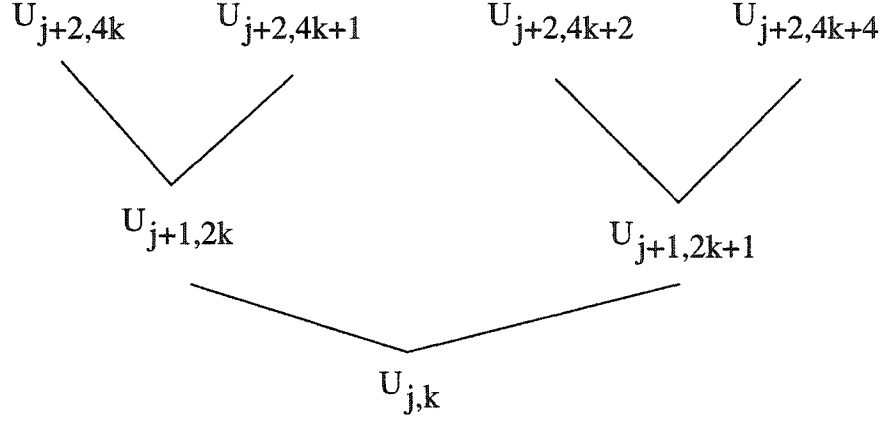


Figure 3.2: Process of generating the scaling coefficients in the DWT.

Now to generate nonnegative data, which in our case represent the stack distance string, we need to have  $|W_{j,k}| \leq U_{j,k}$ . To satisfy this constraint, the wavelet coefficients can be defined as:

$$W_{j,k} = A_{j,k} U_{j,k} \quad (3.10)$$

where  $A_{j,k}$  is a random variable (rv) defined on the interval  $(-1, 1)$ . Using (3.8), (3.9), and (3.10), the following recursion can be obtained for synthesizing the scale coefficients:

$$U_{j+1,2k} = \left( \frac{1 + A_{j,k}}{\sqrt{2}} \right) U_{j,k} \quad (3.11)$$

$$U_{j+1,2k+1} = \left( \frac{1 - A_{j,k}}{\sqrt{2}} \right) U_{j,k} \quad (3.12)$$

The rvs  $A_{j,k}$  must also satisfy the following additional constraints [74]:

1.  $A_{j,k}, k = 0, 1, \dots, 2^j - 1$ , are *i.i.d* rvs that can be represented by the generic rv  $A_j$  having the same CDF as  $A_{j,k}$ .
2. For each  $j$ , the probability density function of the rvs  $A_{j,k}, k = 0, 1, \dots, 2^j - 1$ , is symmetric with zero mean.

3.  $A_j$  is independent of  $A_l$  for  $l > j$  and is also independent of  $U_{0,0}$ .

The wavelet energy at a given scale is given by the variance of the wavelet coefficients at that scale. It has been shown that the correlation structure of the signal can be approximately captured by controlling the wavelet energy decay across scales [74]. The ratio of the energy at scale  $j - 1$  to the one at scale  $j$  ( $j$  is finer than  $j - 1$ ) was found to be [74]:

$$\eta_j = \frac{E[W_{j-1}^2]}{E[W_j^2]} = 2 \frac{E[A_{j-1}^2]}{E[A_j^2](1 + E[A_{j-1}^2])} \quad (3.13)$$

Assuming that  $E[W_j^2]$  is given for all  $j$ , Equation (3.13) can be used to solve for  $E[A_j^2]$ ,  $j = 1, 2, \dots$ . The recursion can be initialized using  $E[A_0^2] = \frac{E[W_0^2]}{E[U_0^2]}$ , where  $W_0$  and  $U_0$  are the wavelet and scale coefficients at the coarsest scale.

In [74], the authors suggested two different distributions for  $A_j$ . One of them is a symmetric beta distribution that has the following pdf:

$$f_{A_j}(x) = \frac{(1+x)^{\rho_j-1}(1-x)^{\rho_j-1}}{\beta(\rho_j, \rho_j)2^{2\rho_j-1}} \quad (3.14)$$

where  $\rho_j$  is the parameter of the rv and  $\beta(\cdot, \cdot)$  is the beta function. The variance of this random variable is given by:

$$\text{var}[A_j] = \frac{1}{2\rho_j + 1}. \quad (3.15)$$

The other distribution is a point-mass distribution defined as:

$$\begin{aligned} \Pr[A_j = c_j] &= \Pr[A_j = -c_j] = r_j \\ \Pr[A_j = 0] &= 1 - 2r_j \end{aligned}$$

In the case of a beta distributed  $A_j$ , the parameter  $\rho_j$  at each scale can be found by solving (3.13) and (3.15), resulting in:

$$\rho_j = \frac{\eta_j}{2}(\rho_{j-1} + 1) - 1/2 \quad (3.16)$$

This, however, assumes that  $E[W_j^2]$  is given for  $j = 1, 2, 3, \dots$ . Since  $\eta_j$ ,  $j = 1, 2, \dots$ , cannot be obtained using a parametric model, it must be computed from

the empirical data, which makes the number of fitted parameters in the model in the order of  $N$ ;  $N$  being the trace length.

On the other hand, if  $A_j$  has a point-mass distribution, then (3.13) by itself is not sufficient to compute both parameters of  $A_j$  ( $c_j$  and  $r_j$ ). An alternative approach for computing these parameters is to rely on the following expression for the moments of the scaling coefficients at different scales:

$$\frac{E[U_j^q]}{E[U_{j-1}^q]} = 2^{-q/2} E[(1 + A_{j-1})^q], \quad q = 1, 2, \dots \quad (3.17)$$

However, to apply (3.17) one needs to have two moments (i.e., two values for  $q$ ) for each scale  $j$ . Again, unless we can compute these values using a parametric model, we have to rely on the empirical data to do so, which makes the model more complex than if a beta distributed  $A_j$  were to be used.

With either distribution of  $A_j$ , it was shown in [74] that the above model generates positive-valued autocorrelated data with an approximately lognormal marginal distribution.

### 3.2.2 Reducing the Number of Parameters

As shown in the previous section, whether  $A_j$  is a beta or a point-mass rv, one needs to provide the second moments of the wavelet coefficients or two moments of the scale coefficients at each scale in order to completely determine  $A_j$ ,  $j = 1, 2, \dots$ . This significantly increases the complexity of the model, as the number of parameters to be computed a priori is in the order of the trace length. Moreover, the point-mass rv is not rich enough and takes only three possible values.

To reduce the complexity of the model, we select  $A_j$  to be a continuous-valued rv with one parameter. Then using (3.17) with  $q = 2$ , we compute the parameter of  $A_j$ ,  $j = 1, 2, \dots$ , assuming that we can compute the ratio  $E[U_j^2]/E[U_{j-1}^2]$  using a small number of parameters (the mean  $\mu$ , the variance  $\sigma$ , and the correlation structure of the modeled data), as shown later. The selection of the rv  $A_j$  will be discussed in Section 3.2.3.

For a discrete time series  $X = \{X_i : i = 1, 2, \dots\}$ , we define  $X^{(m)} = \{X_i^{(m)} : i = 1, 2, \dots\}$  to be the aggregated time series of  $X$  at aggregation level  $m$ :

$$X_n^{(m)} = \sum_{i=nm-m+1}^{nm} X_i, n = 1, 2, 3, \dots, N/m \quad (3.18)$$

where  $m = 1, 2, 4, 8, \dots, N$ ;  $N$  is the length of  $X$ . Note that if the aggregation level  $m$  corresponds to scale  $j$ , then the aggregation level  $2m$  corresponds to scale  $j - 1$ . From the definition of the Haar wavelet transform, the following holds:

$$\frac{E[(X^{(m)})^q]}{E[(X^{(2m)})^q]} = 2^{-q/2} \frac{E[U_j^q]}{E[U_{j-1}^q]}, \quad \text{for } q = 1, 2, \dots \quad (3.19)$$

From (3.19) and (3.17) we get:

$$\frac{E[(X^{(m)})^q]}{E[(X^{(2m)})^q]} = 2^{-q} E[(1 + A^{(2m)})^q] \quad (3.20)$$

where  $A^{(2m)} = A_{j-1}$ . Evaluating (3.20) at  $q = 2$ , we obtain the following expression:

$$E[(A^{(2m)})^2] = 4 \frac{E[(X^{(m)})^2]}{E[(X^{(2m)})^2]} - 1 \quad (3.21)$$

To reduce the number of parameters in the multifractal model, we need to analytically obtain  $E[(X^{(m)})^2]$  for all possible values of  $m$ . The variance at aggregation level  $m$ ,  $\text{var}[X^{(m)}] \triangleq V^{(m)}$ , can be expressed in terms of the autocorrelation function of the signal [23]:

$$V^{(m)} = mv + 2v \sum_{k=1}^m (m - k) \rho_k \quad (3.22)$$

The mean,  $E[(X^{(m)})] = \mu^{(m)}$ , is given by:

$$\mu^{(m)} = m\mu \quad (3.23)$$

where  $\mu$  and  $v$  are the mean and variance of the original signal, respectively. The second moment of  $X^{(m)}$  is then given by:

$$E[(X^{(m)})^2] = mv + 2v \sum_{k=1}^m (m - k) \rho_k + m^2 \mu^2 \quad (3.24)$$



From (3.21) and (3.24), the parameter of the rv  $A_j$  can be computed for all scales  $j = 1, 2, \dots$ , given  $\mu$ ,  $v$ , and the correlation structure of the time series being modeled. For normalized stack distance strings, we found that the form  $\rho_k = e^{-\beta \sqrt[n]{g(k)}}$ ,  $k = 0, 1, \dots$ , fits the correlation structure very well, where  $g$  is a function of the lag  $k$ . For both the ClarkNet and the Worldcup98 traces,  $g(k) = k$  produced a good fit to the empirical ACF, while for the Calgary trace,  $g(k) = \log(k + 1)$  was found appropriate. Figures 3.3, 3.4, and 3.5 show the fitting of the ACF functions for the three traces.

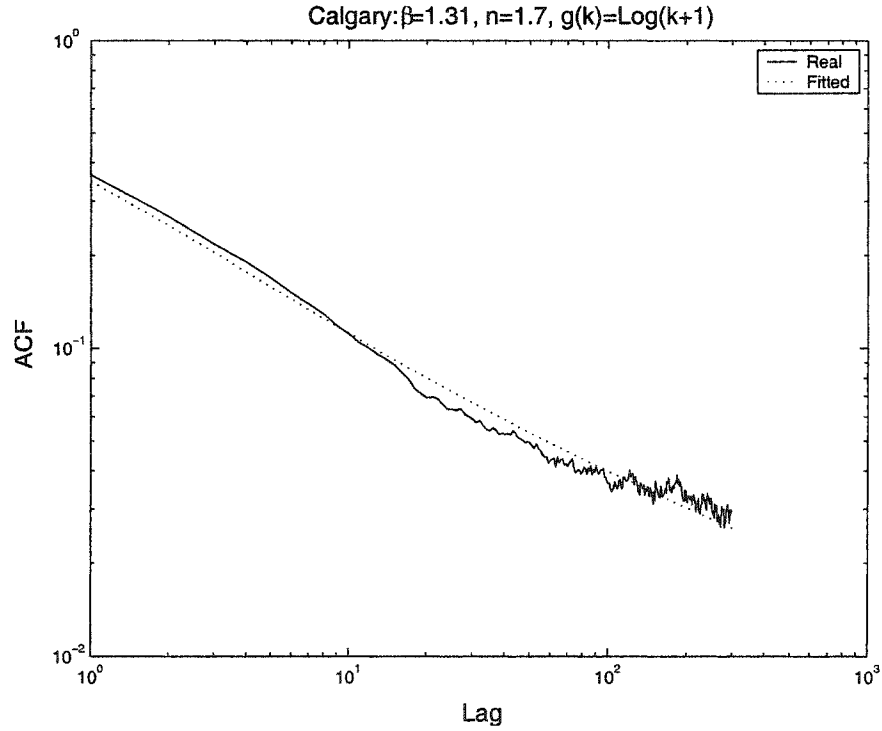


Figure 3.3: Fitting the correlation structure of the normalized stack distance string (Calgary trace).

In summary, to use the multifractal model for modeling the normalized stack distance string, we only need five parameters:

- Mean of the normalized stack distance string ( $\mu$ ).

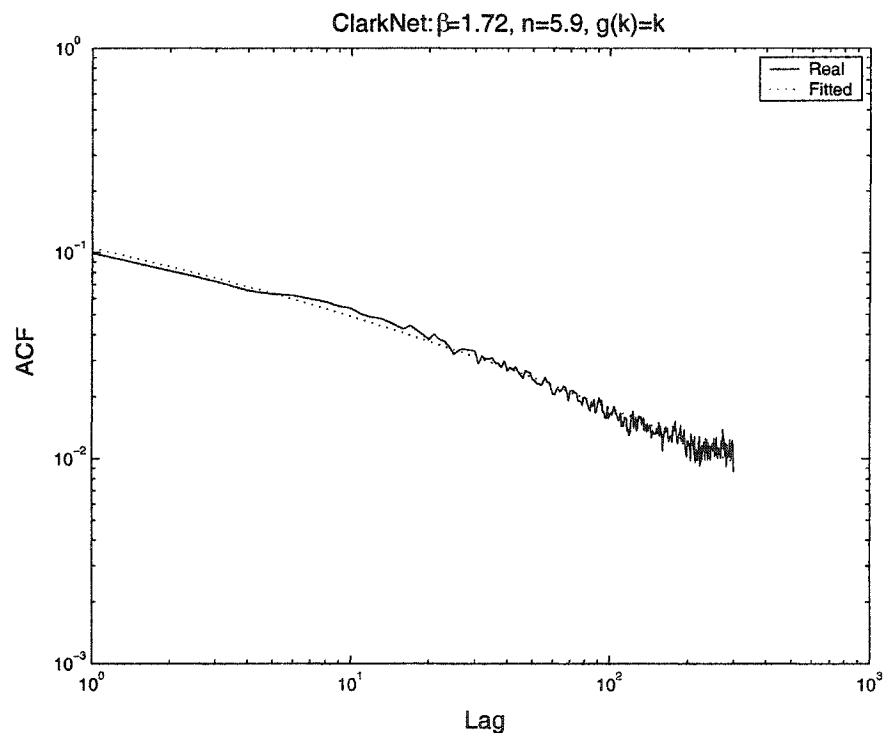


Figure 3.4: Fitting the correlation structure of the normalized stack distance string (ClarkNet trace).

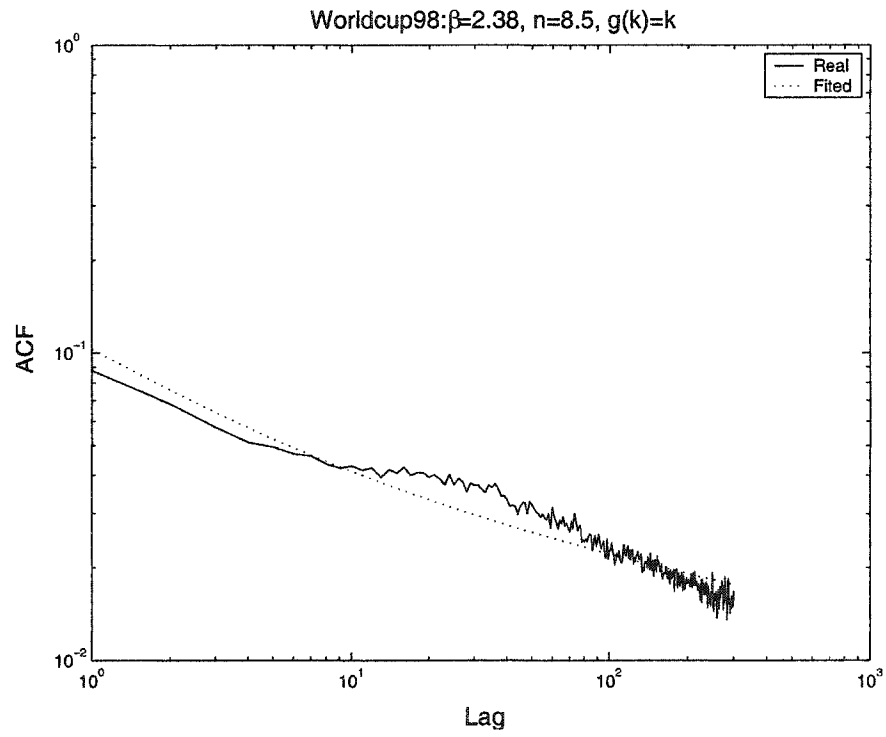


Figure 3.5: Fitting the correlation structure of the normalized stack distance string (Worldcup98 trace).

- Variance of the normalized stack distance string ( $v$ ).
- Autocorrelation structure (parameterized by  $\beta$ ,  $n$ , and  $g$ ).

Using these parameters, along with (3.24) and (3.21), one can compute the parameter of the rv  $A_j$  at each aggregation level (scale).

The synthesis process starts from the highest level of aggregation. At this level we can start with  $l$  data points that are normally distributed with mean  $m_h\mu$  (the mean at aggregation level  $m_h$ ) and variance of  $\text{var}[X^{(m_h)}]$ , where  $m_h$  is the highest aggregation level. After that, the process can be carried out using (3.11) and (3.12).

### 3.2.3 Selecting the Random Variable $A_j$

As indicated earlier, the rv variable  $A_j$  must be symmetric with zero mean and defined on the interval  $(-1,1)$ . To reduce the number of parameters of the multifractal model, we require that the  $A_j$  is specified by one parameter only. There are many rvs that can satisfy these conditions. The difference between one rv and another is the range of the values that the ratio  $E[(X^{(m)})^2]/E[(X^{(2m)})^2]$  can take.

**Theorem 3.2.1** *For any random variable  $X$ , the following hold:*

$$0.25 \leq \frac{E[(X^{(m)})^2]}{E[(X^{(2m)})^2]} \leq 0.5 \quad (3.25)$$

**Proof:** See Appendix A.

As an example, consider the uniform rv in the range  $[-c, c]$ , where  $|c| < 1$ . The variance of this rv is given by  $V = c^2/3$ . Solving (3.21) for  $c$ , we get:

$$c = 3(4 \frac{E[(X^{(m)})^2]}{E[(X^{(2m)})^2]} - 1).$$

Since  $c < 1$ ,

$$\frac{E[(X^{(m)})^2]}{E[(X^{(2m)})^2]} < \frac{1}{3}$$

Using similar calculations, Table 3.1 shows the upper bound for a number of popular rvs.

Random variable	Upper bound
Symmetric beta( $\rho, \rho$ )	0.5000
Uniform( $-c, c$ )	0.3333
Triangular( $-c, 0, c$ )	0.2917
Normal( $0, \sigma$ )	0.2778

Table 3.1: Upper bound on the ratio  $E[(X^{(m)})^2]/E[(X^{(2m)})^2]$  for various distributions.

For the three traces we use in this work, the ratio  $E[(X^{(m)})^2]/E[(X^{(2m)})^2]$  did not exceed 0.3333, and as a result, we decided to use the uniform rv since it is the simplest one.

### 3.3 Stack distance Model

In this section, we describe our stack distance approach for modeling the stream of file objects generated by a WWW server. Let  $U$  be the number of unique files (or objects) at the server and let  $fr_i$  be the fraction of times that the  $i$ th file,  $i = 1, 2, \dots, U$ , appears in the reference string ( $fr_i$  is the popularity profile of file  $i$ ). The modeling approach proceeds in three steps. First, we extract the stack distance string from the URL reference string. Then, we apply some form of normalization to capture both sources of temporal locality (temporal correlation and long-term popularity). The modified multifractal model described in the previous section is then applied to model the normalized stack distance string. Finally, we incorporate the popularity profile of the traffic during the process of generating synthetic reference strings. These steps are described next.

#### 3.3.1 Extracting the Empirical Normalized Stack Distance String

In this model, we use the concept of stack distance to model the temporal and spatial localities in WWW traffic. The authors in [13] extract the stack distances from the original trace assuming an arbitrary initial ordering of the stack. Whenever an object is requested, its depth in the stack (stack distance) is recorded and the object is

pushed to the top of the stack. In our model, we avoid making any assumptions on the initial ordering of the stack, which we have found to disturb the marginal distribution and the correlation structure of the stack distance string. We start with an empty stack and process the empirical reference string *in the reverse direction*, starting from the last reference. If a file is referenced for the first time (in the reverse direction), it is put on top of the stack but no stack distance is recorded. Otherwise, if the file has already been referenced before (hence, it is already in the stack), then it is pushed from its previous location in the stack to the top of the stack and its depth is recorded as a stack distance. Finally, the resulting trace of stack distances is reversed to get the correct stack distance string. The following example illustrates the idea. Consider the reference string [a d c b c d d a b], where each letter indicates the name of a file. If we process this string starting from the end, the first reference is to file b. Since this is the first time file b is being referenced, we push it to the top of the stack without recording any distance. The same procedure is performed for the next two references (for files a and d). The fourth reference (from the end) is for file d. Since this file has been referenced before, it gets pushed to the top of the stack and its stack depth is recorded (in this case, the stack depth for file d is one). The procedure continues until all references are processed (see Figure 3.6). The end result of this process is the stack distance stream [4 3 2 4 1].

Temporal locality in a stream of WWW requests is attributed to two factors: long-term popularity and short-term temporal correlations. Both factors are important for cache design [48, 47], and must therefore be incorporated in the model for temporal locality. In [48], it was found that the (lognormal) distribution of the stack distance string is predominantly affected by the popularity profile (i.e., long-term popularity). So the marginal distribution of the stack distance (without normalization) does not capture the effect of short-term temporal correlations.

To accurately capture the temporal locality of the traffic, we need to isolate the effect of popularity from that of short-term correlations. One solution is to have a separate stack-distance-string model for equally popular objects. Another solution, used in our work, is to capture how much a stack distance deviates from

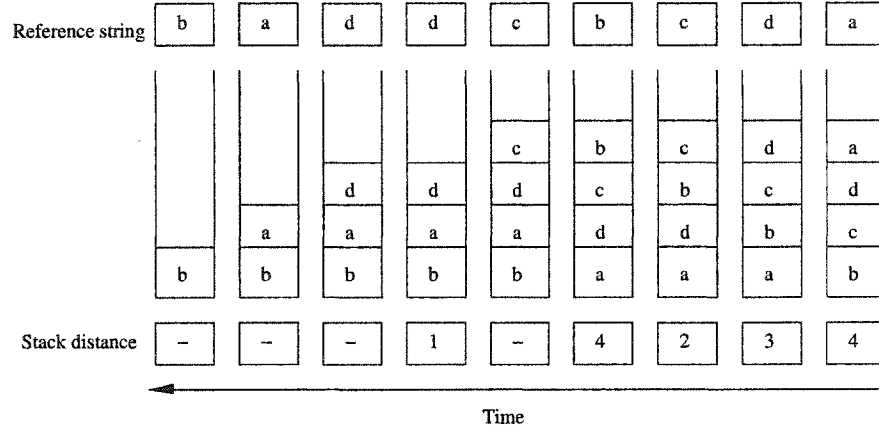


Figure 3.6: Example showing our approach for extracting stack distances from a real trace.

its “expected” value. This deviation is captured by normalizing (scaling) the stack distances by their expected values. The resulting scaled stack distance string is a measure of how files are clustered over the trace length (temporal correlation). This measure is insensitive to the popularity profile, so it allows us to separately model the popularity profile and the short-term temporal correlations.

The expected stack distance for a file  $i$  is computed as follows:

$$E[d_i] = \sum_{g \neq i} \frac{f_g}{f_g + f_i - 1} \quad (3.26)$$

where  $f_i$  is the number of references to file  $i$  [19]. For the three studied traces, we found that the normalized stack distance string has an approximately lognormal marginal distribution. Figures 3.7, 3.8, and 3.9 show the fitting of the lognormal marginal distribution of the normalized stack distance string for the three traces.

### 3.3.2 Modeling the Normalized Stack Distance String

To model the normalized stack distance string, we need to determine  $\mu$ ,  $v$ ,  $\beta$ , and  $n$ . Once the values of these parameters are determined, the multifractal model described in Section 3.2 can be used to capture the marginal distribution and the correlation structure of the normalized stack distance string. Note that spatial locality

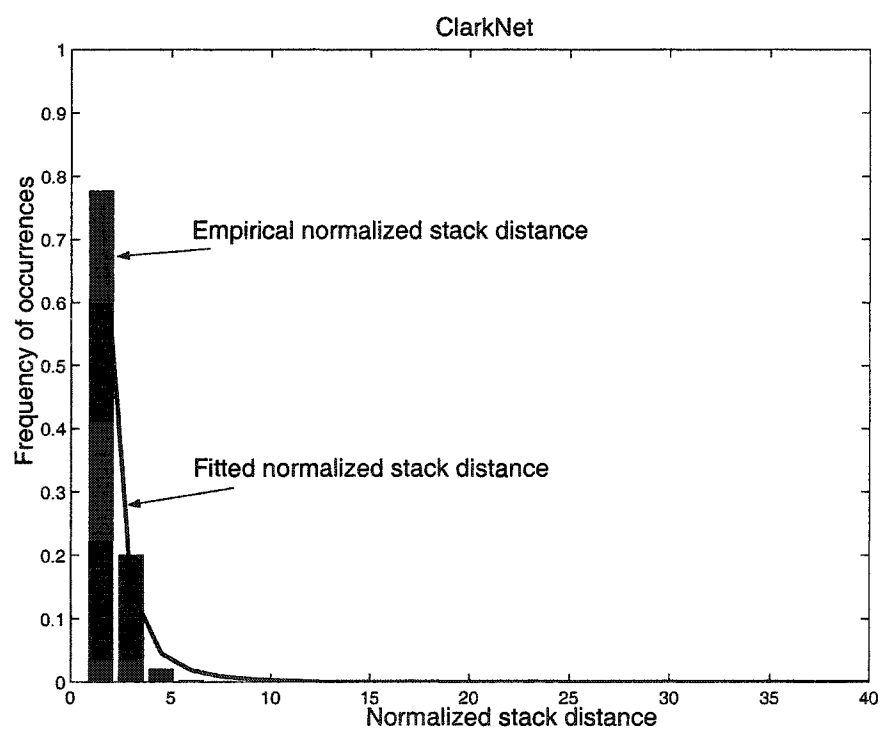


Figure 3.7: Marginal distribution of the normalized stack distance string (ClarkNet trace).



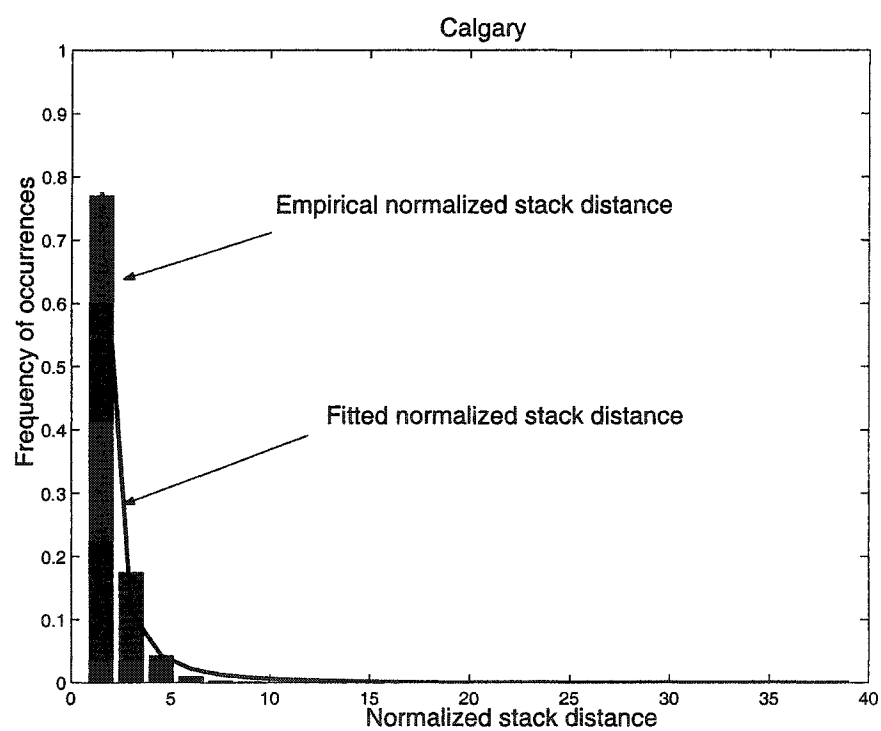


Figure 3.8: Marginal distribution of the normalized stack distance string (Calgary trace).

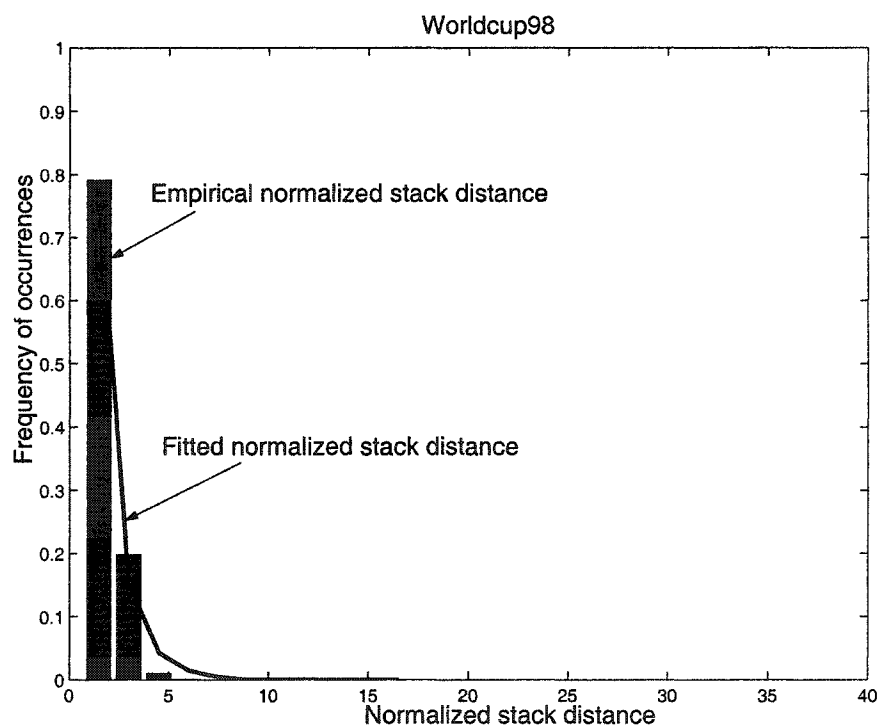


Figure 3.9: Marginal distribution of the normalized stack distance string (Worldcup98 trace).

is captured by modeling the autocorrelation function of the *unscaled* stack-distance string. However, since the multifractal model is applied to the scaled version of the stack-distance string, we have to invert back the synthesized trace of the multifractal model so that the resulting trace models the unscaled version of the data. The pseudo-code in Figure 3.10 describes the modeling process. The function `GenNormStackDistance` takes five input parameters. The first three parameters are  $\mu$ ,  $\nu$ , and the ACF of the normalized stack distance string to be generated. The fourth parameter,  $N$ , is the length of the synthetic normalized stack distance string. The fifth parameter,  $l$ , is the number of data points at the coarsest scale. `GenNormStackDistance` starts by computing the number of aggregation levels, *NumAggLevels* (line 2). In line 4, the second moment of the normalized stack distance string at aggregation level  $m = 1$  is computed. The for loop starting in line 5 is used to compute the second moment for the remaining aggregation levels. The for loop that starts from line 8 computes the summation in (3.24). In line 11, the second moments at higher aggregation levels are computed using (3.24). The second moments ratio,  $E[(X^{(m)})^2]/E[(X^{(2m)})^2]$ , is computed in line 12. The parameter of the rv  $A^{(2m)}$  is computed in line 13 using (3.21) and substituting for the variance of the uniform rv in  $E[(X^{(2m)})^2]$ . After computing these parameters, the mean of the time series at the highest aggregation level is computed in line 15 using (3.23). The variance at the highest aggregation level is computed in line 16. In line 17,  $l$  data points are generated to represent the coarsest level. The while loop starting from line 19 is used to continue the generation process using (3.11) and (3.12).

### 3.3.3 Modeling Popularity and Generating Synthetic Reference Strings

To generate a synthetic WWW reference string, we first generate a synthetic normalized stack distance string, as shown in the previous section. The process of generating a synthetic WWW reference string starts by arranging the unique files of the WWW server in an LRU stack. This is done by sampling from a probability distribution that is weighted by the popularity profiles of the various files (i.e., the more popular a file is, the more likely it will be placed closer to the top of

```

GenNormStackDistance( $\mu, v, \rho_k, N, l$ )
2  NumAggLevels =  $\lceil \log_2(N/l) \rceil$ 
3   $m = 1$ 
4  SecMom(1) =  $mv + m^2\mu^2$ 
5  for  $i = 2$  to NumAggLevels Do
6       $m = 2m$ 
7       $sum = 0$ 
8      for  $k = 1$  to  $m$  Do
9           $sum = sum + (m - k)\rho_k$ 
10     end for
11     SecMom( $i$ ) =  $mv + 2vs + m^2\mu^2$ 
12     SecMom_ratios(NumAggLevels -  $i + 1$ ) =  $\frac{SecMom(i-1)}{SecMom(i)}$ 
13     AParm(NumAggLevels -  $i + 1$ ) =
         $\sqrt{3(4 \text{ SecMom\_ratio}(\text{NumAggLevels} - i + 1) - 1)}$ 
14 end for
15  $\mu_h = 2^{(NumAggLevels-1)} * \mu$ 
16  $V_h = SecMom(NumAggLevels) - \mu_h^2$ 
17 NormStkDist = norm_random( $\mu_h, \sqrt{V_h}, l$ )
18  $i = 1$ 
19 While(length(NormStkDist) <  $N$ ) Do
20      $A = Uniform\_random(-AParm(i), AParm(i), length(NormStkDist))$ 
21     NormStkDist = [NormStkDist  $\frac{(1+A)}{2}$  NormStkDist  $\frac{(1-A)}{2}$ ]
22      $i = i + 1$ 
23 end while
END

```

Figure 3.10: Algorithm for generating synthetic scaled stack distance strings.

the stack). This ordering approach was used in [19]. It is known to provide more accurate results than using an arbitrary ordering. Note that even though the probability of selecting a *given* unpopular file is small, the probability of selecting *any* of the unpopular files is relatively large (because of the large number of unpopular files). So, probabilistically, there is a good chance that *some* unpopular files will be placed near the top of the stack. To generate a reference string of length  $N$ , we first compute the number of references a file can get according to its popularity profile. Then, the top file in the LRU stack is considered as the next referenced file in the synthetic reference string. If the required number of references for this file is reached, then this file is flushed out of the stack. Otherwise, it is pushed down the stack according to the next value in the normalized stack distance string. This is done after scaling back the normalized stack distance by multiplying it by the corresponding expected stack distance for the object in hand (objects with the same popularity profile have the same expected stack distance). Note that our notion of a “stack” allows for the insertion of an object in between two objects in the stack, which does not happen in a regular LRU stack. This process continues until the popularity profiles of all objects are satisfied (no files are left in the LRU stack). The pseudo-code in Figure 3.11 describes the generation process. Function `GenTrace` accepts three parameters: the synthetic normalized stack distance ( $NormStkDist$ ), the number of requests each file gets ( $req$ ), and the  $lru$  stack with the files ordered according to the popularity profile (i.e., placed randomly according to the empirical distribution of the popularity profiles). The while loop in line 4 is used to generate the reference string. Line 5 is used to record the next reference,  $Ref(i)$ , taken as the file at the top of the LRU stack. Then the number of outstanding references to this file,  $req(Ref(i))$ , is reduced by one (line 7). If this number reaches zero, then the file is dropped out the stack. Otherwise, the next stack distance,  $StkDist$ , is computed in line 11 by scaling back the normalized stack distance according to the popularity of the file. The file is then pushed  $StkDist$  positions down the stack. The while loop in line 4 is continued until the LRU stack is empty.

```

GenTrace(NormStkDist, req, lru)
2   i = 0
3   k = 0
4   While(stack is not empty) Do
5       Ref(i) = top(lru)
6       i = i + 1
7       req(Ref(i)) = req(Ref(i)) - 1
8       if (req(Ref(i)) == 0) then
9           drop file from the stack
10      else
11          StkDist = Scale-Back(NormStkDist(k), Ref(i))
12          Push-Top-File-Down-Stack(StkDist)
13          k = k + 1
14      endif
15  end while
END

```

Figure 3.11: Algorithm for generating synthetic WWW strings.

### 3.3.4 Performance Evaluation

In this section, we evaluate the accuracy of the proposed multifractal model and contrast it with two other models. The first model is a self-similar (monofractal) model [4, 13]. This model involves transforming the Gaussian marginal distribution of a fractional ARIMA process into a lognormal distribution. We simply refer to this model as the LRD model. The second model was proposed by Cherkasova et al. [19]. The three investigated models were mainly designed for offline traffic generation, with the primary purpose of generating synthetic traces for use in cache design studies. Accordingly, we compare these models in terms of the file and byte miss ratios seen at an LRU cache that is driven by synthetic traces from these models. The comparison is made with reference to the cache performance seen under the real traffic. As real traffic we use three data sets that were obtained from three separate WWW servers log files: the Computer Science Department WWW server at the University of Calgary, the WWW server at ClarkNet (a commercial Internet

provider in Baltimore, Washington DC), and from the Worldcup98 WWW servers [102]. Table 3.2 provides a summary of the main features of the data sets. More details can be found in [102, 7]. Note that the three traces have contrasting loads (in requests/second). The Calgary's load is the lightest while the Worldcup98's load is the heaviest.

Feature	Trace		
	Calgary	ClarkNet	Worldcup98
Log duration	one year	one week	One day
Start date	Oct 24, 1994	August 28, 1995	May 6, 1998
Log size (MB)	52.3	120.1	107
Total number of requests before reduction	726,739	1,164,868	1,193,353
Total number of requests after reduction	567,519	1,125,092	1,033,567
Number of unique files	8,220	20,168	3,824
Number of files referenced only once	1,752	5,279	665

Table 3.2: Summary of the data sets used in the modeling study.

The data sets contain several pieces of information, including the name of the host that generated the URL request, the day and time the request was recorded, the name of the requested file, the HTTP reply code (explained below), and the number of transferred bytes in response to the request. Four types of HTTP reply codes were recorded: *successful*, *not modified*, *found*, and *unsuccessful*. A *successful* code indicates that the requested file was found at the server and was returned to the client. The client may have a copy of a file, but may want to verify if this copy is up-to-date or not. If the file is up-to-date, the server responds with a *not modified* code. The *found* code indicates that the requested file is available at a different server whose address is provided in the response. Finally, the *unsuccessful* code indicates that the requested file is not available, the client has no permission to access the file, or that there is an error. In our analysis, we only included the requests with *successful* code, since they are the ones that result in actual data transfer from the

server. We also excluded dynamic files (e.g., cgi and pl files). Figures 3.12, 3.13, 3.14, 3.15, 3.16 and 3.17 show the simulation results for the three models.

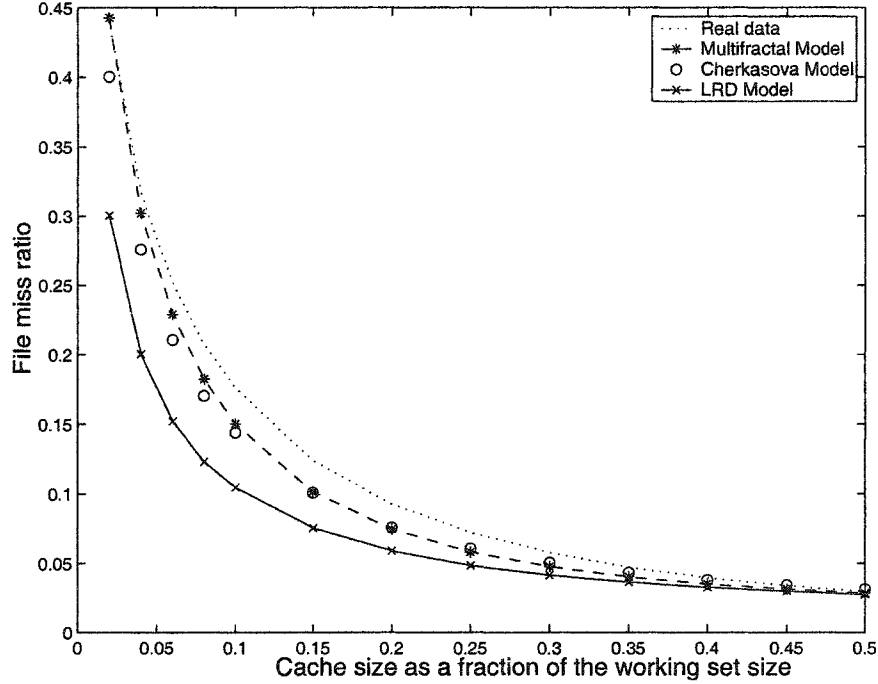


Figure 3.12: File miss ratio versus cache size (Clarknet trace).

It is clear that of the three models, the proposed multifractal model produces the most accurate results, especially for small cache sizes. The relative accuracy in terms of capturing the behavior of the real data is greater in the case of the Calgary data. Consider, for example, the Calgary data with a normalized cache size of 0.3. The percentage inaccuracies in the file miss rate for the multifractal model, the LRD model, and Cherkasova et al.'s model are 0.5%, 53%, and 111%, respectively. In the case of the byte miss rate, the corresponding values are 4.9%, 65%, and 109%. The overall improvement in the accuracy of the file and byte miss rates due to the use of the multifractal model is significant. Moreover, our model captures the spatial locality which is not reflected through the file/byte miss ratios of the LRU caching policy. To show how well our model captures this property, we also compared the inter-request times (number of requested files between any two references to a given



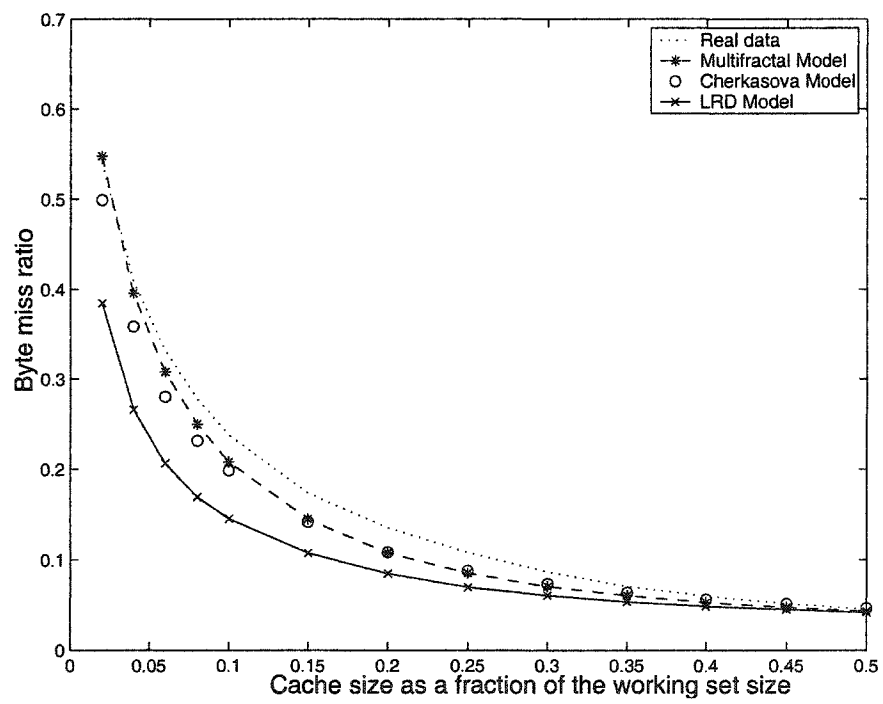


Figure 3.13: Byte miss ratio versus cache size (Clarknet trace).

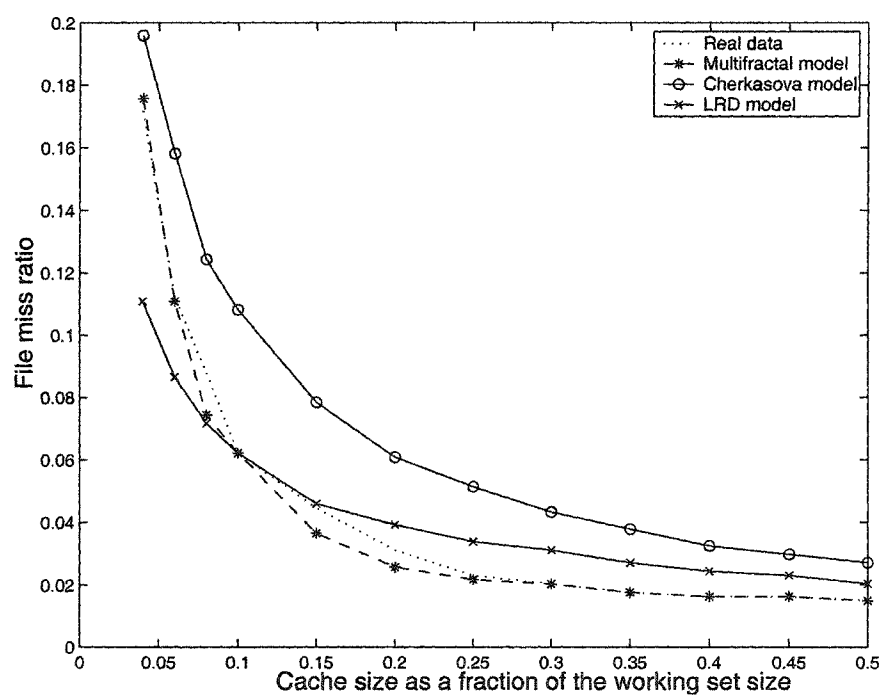


Figure 3.14: File miss ratio versus cache size (Calgary trace).

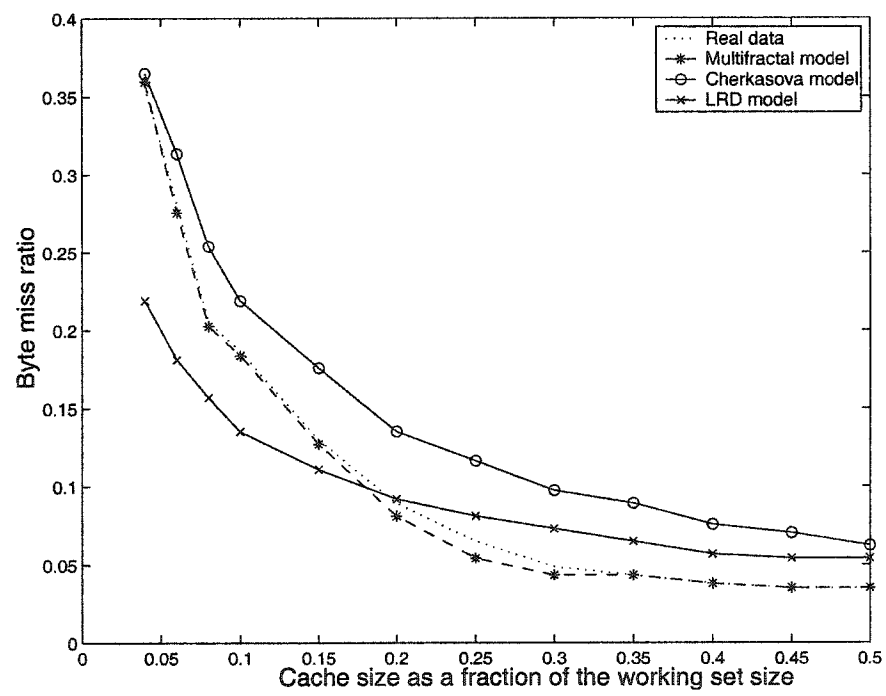


Figure 3.15: Byte miss ratio versus cache size (Calgary trace).

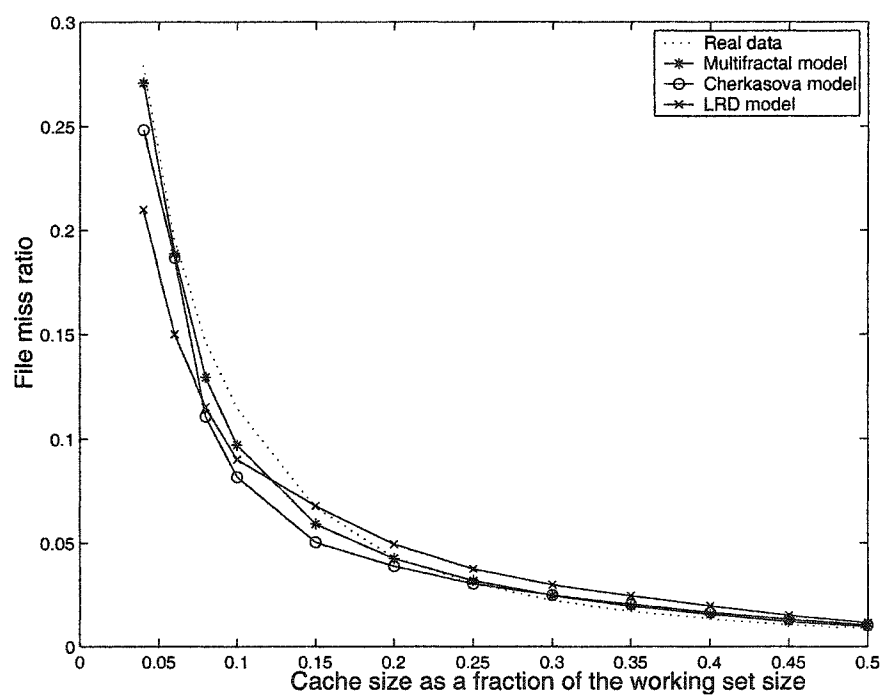


Figure 3.16: File miss ratio versus cache size (Worldcup98 trace).

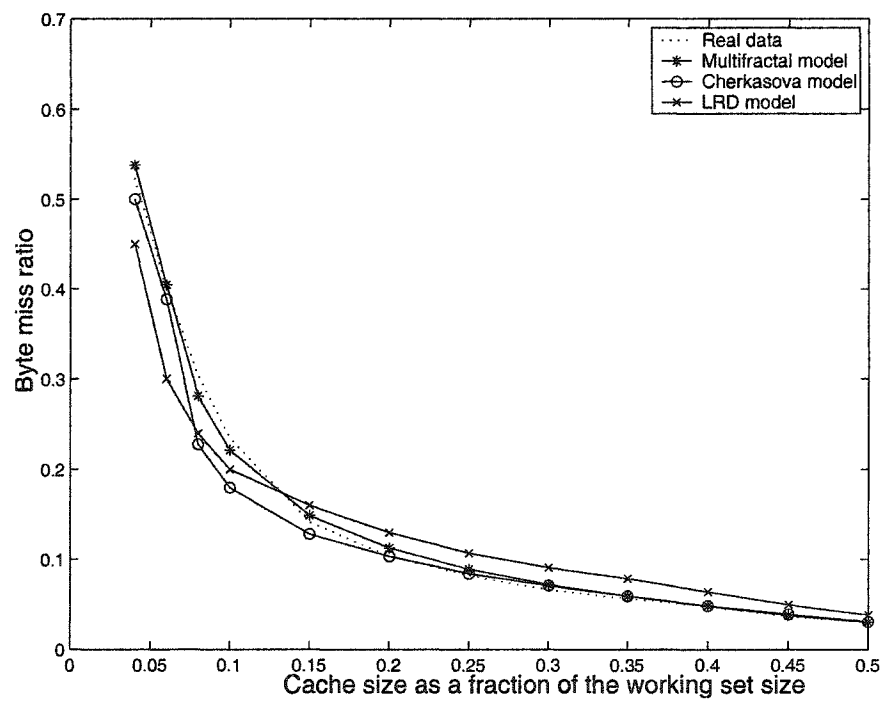


Figure 3.17: Byte miss ratio versus cache size (Worldcup98 trace).

file) in the real data to those in the synthetic traces of the models. We found that the sequence of inter-request times is non-stationary (has a clear decreasing trend). So we normalized this sequence by the *expected* inter-request time, which is equal to  $1/fr_i$ , for file  $i$ . The mean, variance, percentile values, and some values of the autocorrelation function for the normalized inter-request times are shown in Tables 3.3, 3.4, and 3.5. Note that the mean, variance, percentile values of the inter-request times are measures of the temporal locality, while the autocorrelation is a measure of the spatial locality.

Statistics		Real data	Multifractal model	Cherkasova's model	LRD model
$\mu$		0.954	0.937	0.806	0.905
$\sigma$		1.032	1.164	1.918	1.428
$\rho_1$		0.130	0.118	0.000	0.060
$\rho_5$		0.076	0.075	0.000	0.020
$\rho_{10}$		0.061	0.058	0.000	0.009
$\rho_{25}$		0.039	0.039	0.000	0.001
Percentiles	75%	1.306	1.149	0.778	1.063
	90%	2.233	2.210	1.697	2.161
	98%	3.976	4.230	4.980	5.051

Table 3.3: Statistical comparisons for CLARKNET trace.

### 3.4 Inter-request Distance Model

In this model, we capture the main properties of WWW traffic using the inter-request distance string. Analogous to the stack distance model, the marginal distribution of the inter-request distance string can be used as a measure of temporal locality since it measures the closeness in time between requests to the same file. Spatial locality is captured by the inter-request distance correlation structure. To distinguish between temporal correlation and long-term popularity, as was done in the stack distance model, we normalize the inter-request distances by their expected values for the case when the files are uniformly distributed over the whole trace.

Statistics		Real data	Multifractal model	Cherkasova's model	LRD model
$\mu$		0.714	0.68	0.62	0.907
$\sigma$		1.541	2.010	7.080	4.224
$\rho_1$		0.192	0.185	0.000	0.030
$\rho_5$		0.070	0.063	0.000	0.030
$\rho_{10}$		0.036	0.036	0.000	0.027
$\rho_{25}$		0.009	0.010	0.000	0.023
Percentiles	75%	0.813	0.813	0.771	0.326
	90%	1.790	1.790	2.293	2.046
	98%	4.289	5.241	5.531	6.644

Table 3.4: Statistical comparisons for CALGARY trace.

Statistics		Real data	Multifractal model	Cherkasova's model	LRD model
$\mu$		0.990	0.951	0.823	0.941
$\sigma$		0.986	1.137	1.718	1.289
$\rho_1$		0.054	0.054	0.000	0.098
$\rho_5$		0.027	0.037	0.000	0.089
$\rho_{10}$		0.025	0.030	0.000	0.071
$\rho_{25}$		0.022	0.023	0.000	0.034
Percentiles	75%	1.373	1.136	0.856	1.134
	90%	2.266	2.099	1.727	2.101
	98%	3.843	4.210	4.666	4.726

Table 3.5: Statistical comparisons for WORLDCUP98 trace.

The expected value of the inter-request distance for a file  $p$  ( $IR(p)$ ) in the uniform case is defined as:

$$E[IR(p)] = \frac{1}{fr_p} \quad (3.27)$$

where  $fr_p$  is the popularity of file  $p$ .

The marginal distribution of the normalized inter-request distance string was found to have the same distribution as the normalized stack distance string (log-normal distribution). The correlation structure model of the normalized stack distance string,  $\rho_k = e^{-\beta \sqrt[k]{g(k)}}$ , can be used for the normalized inter-request distance string too.

After extracting the normalized inter-request distance string from the empirical reference string, we compute its mean  $\mu$ , variance  $\sigma^2$ , and fit its correlation structure. These parameters directly capture a real workload, as opposed to the parameters of the stack distance string. Using these parameters, the multifractal model can then be applied to generate normalized inter-request distances.

#### 3.4.1 Traffic Generation Process

To generate a synthetic reference string, we first need to model the inter-arrival distance of new files (for simplicity we only consider the marginal distribution). As expected, this marginal distribution was found to follow a log-normal like distribution. The process of generating synthetic reference string starts by arranging the unique files in a vector according to their popularity, as we do in the stack distance model. After sampling from the distribution of the inter-arrival distance of new files, we assign each file in the vector a location (real value). This is the location of the first occurrence of each file in the generated trace. The first file is assigned a location value of zero. Knowing the popularity profile and the trace length, we compute the number of requests each file gets. Next, we scan the vector starting from the first file. If the file reaches the required number of requests, we skip it and go to the next file in the vector. Otherwise, the location of the next request to the file is computed as the current reference location plus the next synthetic inter-request



distance. This is done after scaling the normalized inter-request distance back by multiplying by the file's expected inter-request distance. Another reference to this file is inserted into the vector according to the new location. The process continues until all files reach their required numbers of requests. The example in Figure 3.18 illustrates the process. In this example, we assume that we have four unique files, a, b, c, and d, with popularity profile  $\{(a,0.1), (b,0.2), (c,0.3), (d,0.4)\}$ . Suppose we want to generate a synthetic trace of length 10. We first order these files in a vector according to their popularity profile (the more popular a file is, the more likely it will be placed close to the top of the vector). Let the initial order be  $[d \ b \ c \ a]$ . Sampling from the distribution of the new files inter-arrival distance, we assign the locations of the first occurrences of the files in the trace. We assign the first file in the vector a position 0.0. Suppose that the new files inter-arrival distance samples are  $\{3.4, 3.4, 3.6\}$ . Then file d takes position 0.0, file b takes position  $0.0 + 3.4 = 3.4$ , file c takes position  $3.4 + 3.4 = 6.8$ , and file a takes position  $6.8 + 3.6 = 10.4$ . Now we generate a normalized inter-request distance synthetic string using the multifractal model. Let this string be  $\{3.8, 1.34, 1.05, 0.28, 1.4, 1.53\}$ . Using (3.27), we compute the expected inter-request distance for all unique files, which in our example are found to be  $\{(a,10.0), (b,5.0), (c,3.33), (d,2.5)\}$ . Scanning the vector from the top, we find that file d is on the top of the vector with a position 0.0. The first normalized inter-request distance is 3.8. Scaling this value back by multiplying by the expected inter-request distance for file d, we can compute the position of the next reference to file d, which is equal to  $0.0 + 3.8 * 2.5 = 9.5$ . A new reference to file d is inserted in position 9.5. The process continues until all the required number of references for all files are satisfied. Once a file reaches its required number of references, it is skipped (as can be seen in the fifth column from the left in Figure 3.18, where file b is skipped).

### 3.4.2 Performance Evaluation

In this section, we evaluate the performance of the inter-request distance (IRD) model, and compare it with the stack distance (SD) model. Both models are mainly

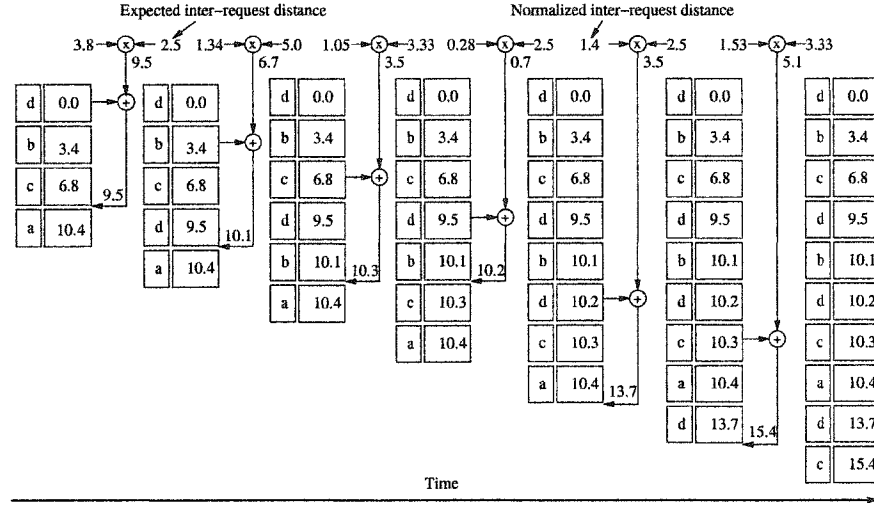


Figure 3.18: WWW synthetic trace generation process.

intended for offline generation of WWW synthetic traces. We compare the two models in terms of the file miss ratio seen by an LRU cache that is driven by synthetic traces from both models. Moreover, since the spatial locality is not reflected through the file miss ratios of the LRU caching policy, we also compare both models in terms of the mean, variance, correlation, and percentile values of the inter-request distance string of the synthetic traces. We found that the inter-request distance string exhibits a clear increasing trend and for this reason we used the normalized one in the comparison. The real traffic is considered as a reference for the comparison. Tables 3.6 and 3.7 show the inaccuracies in the LRU miss ratios resulting from two traces (relative to the cache miss ratio for the real trace). The cache size is measured as a percentage of the total size of the unique files. It is clear that the IRD model is more accurate than the SD model, which is an indication of a better capturing of the temporal locality of the real trace. For example, in the worldcup98 trace with a cache size of 10%, the percentage of inaccuracies in the file miss rate for the IRD model and the SD model are given by 5.34% and 15.6% respectively. Tables 3.8 and 3.9 show the inaccuracies in the mean, variance, correlation, and in some percentile values of the normalized inter-request distance string from both traces. Again, it is clear that the IRD model is more accurate than the SD model in capturing these

statistics, which is an indication of better capturing of the spatial locality of the real trace.

Cache Size (%)	Real data Miss Ratio (%)	SD Model Error (%)	IRD Model Error (%)
4%	27.9	3.02	3.1
6%	19.6	3.61	3.6
8%	14.7	11.8	5.25
10%	11.5	15.6	5.34
15%	6.70	11.8	7.57
20%	4.32	1.94	2.1
25%	3.05	4.54	4.54
30%	2.23	10.3	6.19
35%	1.71	14.2	5.91
40%	1.35	15.5	5.68
45%	1.07	13.3	8.33
50%	0.88	9.69	6.93

Table 3.6: Cache miss ratio inaccuracy for the worldcup98 trace.

Cache Size (%)	Real data Miss Ratio (%)	SD Model Error (%)	IRD Model Error (%)
4%	31.7	4.71	1.21
6%	25.2	9.05	3.97
8%	20.8	12.0	3.41
10%	17.6	14.7	4.79
15%	12.4	18.2	6.3
20%	9.26	19.4	6.77
25%	7.22	19.4	6.27
30%	5.77	17.7	4.35
35%	4.72	14.7	0.81
40%	3.96	11.4	3.33
45%	3.39	7.66	2.31
50%	2.96	3.78	3.5

Table 3.7: Cache miss ratio inaccuracy for ClarkNet trace.

Statistics		Real data	SD Model Error (%)	IRD Model Error (%)
$\mu$		0.990	3.94	0.81
$\sigma$		0.986	15.3	0.51
$\rho_1$		0.054	0.00	1.85
$\rho_5$		0.027	37.0	25.9
$\rho_{10}$		0.025	20.0	8.00
Percentiles	75%	1.373	17.3	3.71
	90%	2.266	7.37	1.37
	98%	3.843	9.55	1.74

Table 3.8: Statistical comparisons for the worldcup98 trace.

Statistics		Real data	SD Model Error (%)	IRD Model Error (%)
$\mu$		0.954	1.78	0.52
$\sigma$		1.032	12.8	0.68
$\rho_1$		0.130	9.23	1.54
$\rho_5$		0.076	1.32	1.32
$\rho_{10}$		0.061	4.92	0.00
Percentiles	75%	1.306	12.0	8.04
	90%	2.233	1.03	0.54
	98%	3.976	6.39	0.55

Table 3.9: Statistical comparisons for ClarkNet trace.

## CHAPTER 4

### WWW PREFETCHING MODEL

#### 4.1 Introduction

Aggressive or Non-controlled prefetching was found to degrade the performance by increasing the load in the system, which in turn may worsen the WWW client perceived latency. Accordingly, in this chapter we investigate the trade off between the reduction in the response time and the increase in the system load due to prefetching. The aim of such investigation is to come up with a dynamically controlled prefetching protocol that can balance this trade off and optimize the performance. Such protocol can dynamically adjust the degree of prefetching (how much to prefetch) according to the state of the system. We achieve this goal through a mathematical model that incorporates the system essential parameters to characterize both effects of prefetching.

The rest of the chapter is organized as follows. In Section 4.2 we describe the prefetching system architecture. In Section 4.3 we describe our network access model and derive an expression for the prefetching gain as a function of the system parameters (system load and cache parameters). In Section 4.4, we treat the prefetching problem as an optimization problem, where we compute a threshold value for the prefetching precision. Moreover, we answer the question of “How many documents to prefetch?” In Section 4.5, we describe the use of our model in designing a prefetching protocol. In Section 4.6, we discuss the effects of proxy caching and local caching on prefetching performance. In Section 4.7 we validate our model through simulations.

#### 4.2 Prefetching System Architecture

Consider a group of WWW clients who are connected to a proxy server through dedicated lines (i.e., dial-up modems, cable modems, DSL, etc.) of capacity  $r$  bits

per second (see Figure 4.1). The proxy server is connected to the Internet via an access link of capacity  $C$  bps. A client is assumed to run one browsing session at a time. The case of multiple sessions will be treated in a future work. Each client maintains a local cache. To generalize the treatment, an arbitrary cache replacement policy is assumed and is parameterized by its hit ratio  $h_c$ . A very small portion of the client cache is reserved for prefetching, and is called the *prefetching cache*. The remaining portion is called the *regular cache*. It was reported in several studies (e.g., [4, 18, 27, 17]) that the hit ratio increases as the logarithm of the cache size. Hence, reserving a small portion of the cache for prefetching has a negligible effect on the hit ratio of the regular cache, making this hit ratio almost independent of prefetching. The regular cache stores demand-requested documents, whereas the prefetching cache stores prefetched documents. A demand-requested document that happens to be in the prefetching cache is considered for regular caching (i.e., is moved to the regular cache). Accordingly, a given document cannot be in both caches at the same time. Prefetched documents are brought to the local cache from either the proxy server (if available) or are retrieved from the original WWW server. The performance of the proxy cache is described by its hit ratio  $h_{proxy}$ , which is assumed to be independent of prefetching (the proxy does not cache any prefetched file). We verify this point later in the simulations. Each client alternates between active (ON) periods, during which the client retrieves some documents, and idle (OFF) periods, during which the retrieved information is read by the user (see Figure 4.2). An ON period starts with the retrieval of an html file (the *main document*), which is usually followed by the retrieval of its inline files.

Each client runs a prediction algorithm that predicts future requests using the history of the client's requests and hints from the proxy and original servers. We assume that the HTTP protocol can be easily modified to include these hints in the header of the response to any regular request. Typically, the outcome of the prediction algorithm becomes available right after the receipt of the main document. We assume a generic prediction model, where the predictor computes a set of  $k$  candidate files  $D_1, D_2, \dots, D_k$  and the probabilities of accessing them in the next

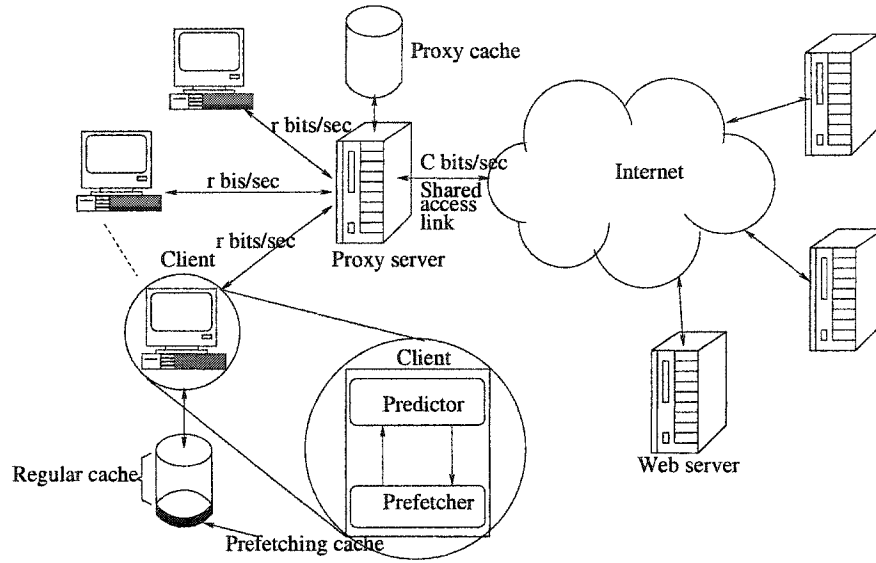


Figure 4.1: Components of the prefetching system.

user's active period  $(P_1, P_2, \dots, P_k)$ . For example, one can adopt the scheme in [45] with a straightforward modification to account for hints from the proxy server (the details of such modification will be described later). Note that the events of requesting any two or more files in an ON period are not necessarily mutually exclusive, i.e.,  $\sum_{i=1}^k P_i$  can be greater than one. The prefetcher uses the information provided by the predictor to prefetch files in the subsequent OFF period, starting from the one with the highest access probability. The number of prefetched files depends on the length of the OFF period and the state of the network. If the OFF period is large enough, prefetching ends before the start of the next ON period. Otherwise, the client instructs the proxy to stop forwarding the currently prefetched file once a new *demand-request* (new ON period) is issued. Any partially prefetched file is kept in the prefetching cache to be used in any future access to such a file. A demand-request is first served from the local cache (regular or prefetching cache) if available. Otherwise, the request is forwarded to the proxy server. If the proxy server does not have the requested file in its cache, it retrieves it from the original server.



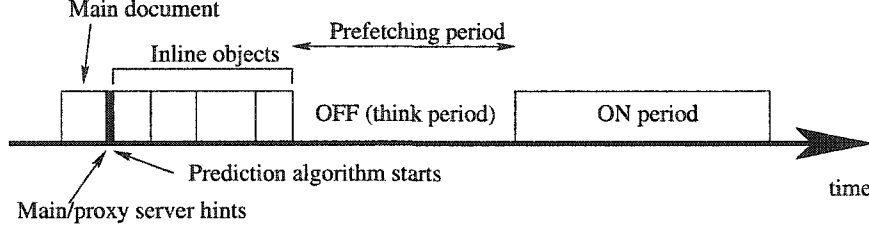


Figure 4.2: Client behavior.

### 4.3 Analysis of Prefetching Gain

In this section, we study the benefit of client-side prefetching using access delay as the performance metric. The improvement in the access delay is indicated by the ratio of the average access time of an arbitrary demand-requested file <sup>1</sup> under prefetching ( $A_p$ ) to the average access time of such a file without prefetching ( $A_{np}$ ). We call this ratio *the access improvement index* ( $I$ ). Prefetching is advantageous when  $I < 1$ . We study the effectiveness of prefetching in reducing the average access time for a given client. In the absence of client caching and prefetching, the proxy server is assumed to retrieve files from the original servers at a rate  $\lambda$  (in files per second) in response to requests from all clients. Note that caching and prefetching can impact the rate of bringing files from their respective servers.

Prefetching always increases the hit ratio of the client cache because prefetched files do not replace any cached ones (they are stored in the prefetching cache). Suppose that, on average, a client prefetches  $N_p$  files in a given OFF period. Then, the average number of “useful” files is:

$$m = N_p \bar{P} \quad (4.1)$$

where  $\bar{P} \triangleq \frac{\sum_{i=1}^{N_p} P_i}{N_p}$  ( $0 \leq \bar{P} \leq 1$ ) is the *prefetching precision* [73]. The increase in the client-cache hit ratio due to prefetching is given by:

$$\Delta_h = \frac{m}{N_{on}} \quad (4.2)$$

---

<sup>1</sup>A web document consists of one or more files.

where  $N_{on}$  is the average number of files in an ON period. This says that for each demand-requested file, there are  $\frac{m}{N_{on}}$  useful prefetched ones.

If a client does not employ prefetching, a requested file is brought from the local cache, the proxy cache, or the original server. The corresponding access times for a file of an average size  $\bar{s}$  are 0,  $t_{prox}(\bar{s})$ , and  $t_{serv}(\bar{s})$ , respectively. Hence, the average access time without prefetching for a file of size  $\bar{s}$  is

$$A_{np} = (1 - h_c) \cdot (h_{proxy} t_{prox}(\bar{s}) + (1 - h_{proxy}) t_{serv}(\bar{s})). \quad (4.3)$$

When prefetching is employed, the access time from the original server is denoted by  $t'_{serv}(\bar{s})$ . Note that  $t'_{serv}(\bar{s}) \neq t_{serv}(\bar{s})$  because prefetching files for a given client increases the traffic seen by other clients that share the same access link, which as a result affects the given client. The time to retrieve a demand-requested file from the proxy cache is not affected by prefetching, because prefetching is done in the OFF period of that client, who communicates with the proxy via a dedicated line. Accordingly,

$$A_p = (1 - h_c - \Delta_h) \cdot (h_{proxy} t_{prox}(\bar{s}) + (1 - h_{proxy}) t'_{serv}(\bar{s})). \quad (4.4)$$

From (4.3) and (4.4), the access improvement index becomes:

$$I = \frac{(1 - h_c - \Delta_h) \cdot (h_{proxy} t_{prox}(\bar{s}) + (1 - h_{proxy}) t'_{serv}(\bar{s}))}{(1 - h_c) \cdot (h_{proxy} t_{prox}(\bar{s}) + (1 - h_{proxy}) t_{serv}(\bar{s}))}. \quad (4.5)$$

We assume that  $t_{serv}(\bar{s})$  and  $t'_{serv}(\bar{s})$  are dominated by the queueing/service delays at the shared access link between the proxy server and the Internet (downlink direction). This assumption is justified when the pool of clients that share the access link is large, as is often the case in ISP networks. To compute  $t_{serv}(\bar{s})$  and  $t'_{serv}(\bar{s})$ , we model the queueing/service delays at the proxy using an M/G/R Processor Sharing (M/G/R-PS) system. Riedl et al. [77] suggested the use of this model for the dimensioning of IP access networks with elastic traffic and concluded its suitability for WWW delivery systems, particularly when file sizes are large. The rationale behind employing the M/G/R-PS approximation is that in the underlying WWW delivery system, multiple file downloads occur simultaneously over different

connections (clients). These downloads are serviced by a shared link (processor) of capacity  $C$ . In our case, clients can be limited by the bandwidth  $r$  of the *dedicated* access link, which can be less than  $C$ . A special case of the M/G/R-PS is when  $R = 1$ . In this case, a single client can fully utilize the capacity of the shared access link. For a client's peak rate  $r$ , the shared link behaves approximately as a system with  $R = C/r$  servers. If there are  $n$  customers in the system, then each customer gets a fraction of the capacity  $C$  that depends on  $n$ ; if  $n \leq R$ , then each customer gets a fixed fraction  $r/C$ . Otherwise, each customer gets a fraction  $1/n$ . This means that up to  $R$  flows can be served simultaneously, each at rate  $r$  bps.

For the M/G/R-PS model, the mean file transfer time is given by [77]:

$$\bar{t} = \frac{\bar{s}}{r} f_R(\rho) \quad (4.6)$$

where  $\bar{s}$  is the file average size and

$$f_R(\rho) \triangleq 1 + \frac{E_2(R, \rho)}{R(1 - \rho)} \quad (4.7)$$

$$E_2(R, \rho) \triangleq \frac{\frac{(R\rho)^R}{R!} \frac{1}{1-\rho}}{\sum_{i=0}^{R-1} \frac{(R\rho)^i}{i!} + \frac{(R\rho)^R}{R!} \frac{1}{1-\rho}}. \quad (4.8)$$

Equation (4.8) is the Erlang C formula. The utilization of the access link is given by  $\rho = \lambda \bar{s} / C$ . The quantity  $f_R$  is called the *delay factor*. It is a measure of how link congestion affects the response time.

To apply the above model, we need to compute the average load on the shared access link (downward direction) for the prefetching and no prefetching cases. The average load in the case of no prefetching (with caching only) is given by:

$$\rho_{np} = \frac{(1 - h_{proxy})(1 - h_c)\lambda \bar{s}}{C}. \quad (4.9)$$

This represents the downlink traffic in response to client requests that cannot be satisfied from either the regular cache or the proxy cache.

When prefetching is implemented, an average of  $N_p$  files are retrieved during the OFF period. Hence, the average load on the shared access link under prefetching is

given by:

$$\rho_p = \frac{(1 - h_{proxy})(1 - h_c - \Delta_h + N_p/N_{on})\lambda\bar{s}}{C}. \quad (4.10)$$

This is the load on the downlink in response to requests that cannot be satisfied from the regular, the prefetching, or the proxy caches, plus the extra prefetched traffic ( $\frac{N_p\lambda\bar{s}}{N_{on}}$ ). Note that for each demand-requested file, there are on average  $N_p/N_{on}$  prefetched ones.

Because each client performs prefetching during its idle (think) periods and only one browsing session is allowed per client, the queuing delay at the (dedicated) proxy-client link can be safely ignored. Hence, the average time to retrieve an arbitrary file of size  $\bar{s}$  from the proxy server is  $t_{prox}(\bar{s}) = \frac{\bar{s}}{r}$ .

From (4.5) and (4.6), the improvement index reduces to:

$$I = \frac{(1 - h_c - \Delta_h) \cdot (h_{proxy} + (1 - h_{proxy})f_R(\rho_p))}{(1 - h_c) \cdot (h_{proxy} + (1 - h_{proxy})f_R(\rho_{np}))}. \quad (4.11)$$

#### 4.4 Prefetching Gain Optimization

In this section, we study the performance of a generic prefetching system. We use the analysis in Section 4.3 to optimize the effect of prefetching. Intuitively, there are two factors that affect the mean access time for a file. On the one hand, prefetching more files improves the overall hit ratio and, as a result, reduces the number of files that need to be retrieved from the original server. On the other hand, prefetching more files results in increasing the load on the shared access link, which affects the retrieval time of other missed files (which are not in the cache nor are being prefetched). Hence, a client should be careful not to prefetch every file suggested by the predictor, as this may lead to increasing the overall average access time.

Accordingly, we seek to compute the optimal number of files to prefetch in an OFF period. Before trying to find this optimal value, we need to study the behavior of  $I$  as a function of  $N_p$ . It can be mathematically shown (see below) that if prefetching a single file or a fraction of a file does not lead to any gain, then prefetching more files can only worsen the performance. On the other hand, if there is a gain out of prefetching a single file or a fraction of a file, then there is a unique optimal

value for the average number of prefetched files in an OFF period. The following theorem describes the general relationship between  $I$  and  $N_p$ . It also specifies the condition under which prefetching is beneficial.

**Theorem 4.4.1** *Suppose that files are prefetched in a decreasing order of their access probabilities, starting from the most likely one. Then the following hold:*

1. *If prefetching a single file or a fraction of a file does not improve the mean file access time, then increasing the number of prefetched files does not do any better.*
2. *If there is a gain out of prefetching, then there is only one optimal value for the average number of prefetched files in an OFF period.*
3. *For prefetching to be of a value, the following condition must be satisfied:*

$$\bar{P}(N_p = 1) > \frac{M\rho_{np}^R}{(1 + M\rho_{np}^R)} \triangleq P_{th} \quad (4.12)$$

where

$$M \triangleq \frac{(1 - h_{proxy})R}{(1 - \rho_{np}^R)(1 - h_{proxy}\rho_{np}^R)} \quad (4.13)$$

and  $\bar{P}(N_p = 1)$  is the prefetching precision when, on average, only one document is prefetched in an OFF period. In other words,  $\bar{P}(N_p = 1)$  is the average access probability of the first file to prefetch in the list of candidate files.

**Proof:** See Appendix B.

It is clear from Theorem 4.4.1 that a prefetching protocol must first decide whether to prefetch or not based on the prefetching condition (threshold). After that, if prefetching is beneficial, then the optimal number of files to prefetch can be computed. To find this optimal value, we need to solve the equation  $\frac{dI}{dN_p} = 0$  for  $N_p$ . Unfortunately, such an equation cannot be analytically solved, except for some special cases, as shown in the rest of this section. In general, one can rely on numerical methods to solve for the optimal  $N_p$ , which we denote as  $N_p^*$ .

#### 4.4.1 Prefetching Precision Independent of $N_p$

Consider the special case when  $\bar{P}$  is independent of  $N_p$  (i.e., all files have the same access probabilities). Accordingly, the condition in (4.12) translates into having the file access probability greater than  $P_{th}$ . In this case,  $N_p^*$  can be computed analytically for two special cases, as described in the following theorem.

**Theorem 4.4.2** *Consider the case when  $\bar{P}$  is independent of  $N_p$ ,  $\bar{P} > P_{th}$ , and  $R = 1$ . Then,*

$$1. \quad N_p^* = \begin{cases} \frac{-B - \sqrt{B^2 - 4AC}}{2A}, & \text{if } B^2 - 4AC > 0 \\ \text{Largest number of candidate} & \\ \text{files subject to } \rho_p < 1, & \text{otherwise} \end{cases} \quad (4.14)$$

where

$$A \triangleq \left( \frac{\rho_{np}(1 - \bar{P})}{N_{on}(1 - h_c)} \right)^2 \quad (4.15)$$

$$B \triangleq \frac{-2\rho_{np}(1 - \bar{P})(1 - \rho_{np})}{(1 - h_c)N_{on}} \quad (4.16)$$

$$C \triangleq (1 - \rho_{np})^2 - \frac{1 - h_{proxy}}{h_{proxy}} \left( \frac{\rho_{np}}{\bar{P}} - 1 \right) \quad (4.17)$$

2. *If no proxy caching is used ( $h_{proxy} = 0$ ), then the higher the number of prefetched files, the higher is the prefetching gain.*

**Proof:** See Appendix C.

Figure 4.3 depicts  $I$  as a function of  $N_p$  for the special case when  $\bar{P}$  is constant,  $R = 1$ , and  $h_{proxy} = 0$ . It is clear that when  $\bar{P} > P_{th}$ ,  $I$  decreases monotonically with the number of prefetched documents. In this case,  $I$  can be maximized by prefetching all documents with access probabilities greater than  $P_{th}$  [84].

For the other cases,  $I$  does not necessarily decrease monotonically with the number of prefetched files. This is shown in Figure 4.4-a ( $R = 10$  and  $h_{proxy} = 0$ ) and Figure 4.4-b ( $R = 1$  and  $h_{proxy} = 0.6$ ). It can be seen from Figure 4.4-a that when  $\bar{P} = 0.4$ ,  $I$  decreases with the increase in  $N_p$  up to a certain point, after which the

trend is reversed. Furthermore, when  $\bar{P} \gg P_{th}$ , the trend in the access improvement becomes monotone. This is because the improvement in the hit ratio is more significant than the loss due to the increased traffic. Moreover, prefetching cannot go beyond a point where the shared access link is 100% loaded.

Note that the threshold value decreases with the increase in  $R$  and  $h_{proxy}$ , which is intuitive since increasing  $R$  or  $h_{proxy}$  moves the delay bottleneck towards the client-proxy link.

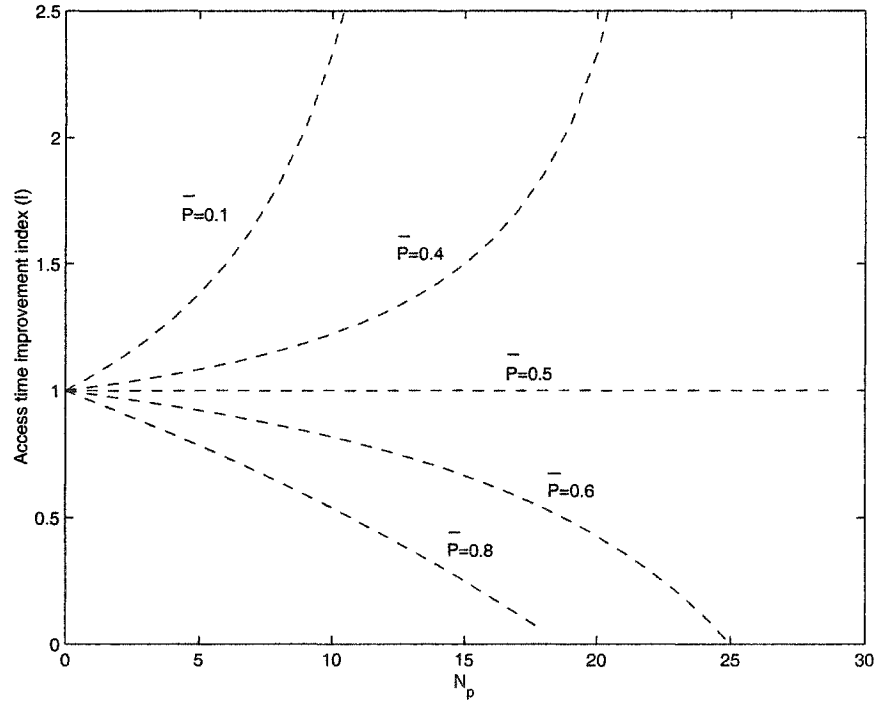


Figure 4.3:  $I$  versus  $N_p$  ( $r = 500$  kbps,  $C = 500$  kbps,  $\lambda = 6.25$  files/s,  $\bar{s} = 40$  kbits,  $N_{on} = 15$  files,  $P_{th} = 0.5$ ,  $h_c = 0$ ,  $h_{proxy} = 0$ ).

#### 4.4.2 Prefetching Precision Varies with $N_p$

To study the effect of the variability of  $\bar{P}$  on  $I$ , consider the following simple relationship between  $\Delta_h$  and  $N_p$ :

$$\Delta_h = K (1 - e^{-a N_p}) \quad (4.18)$$

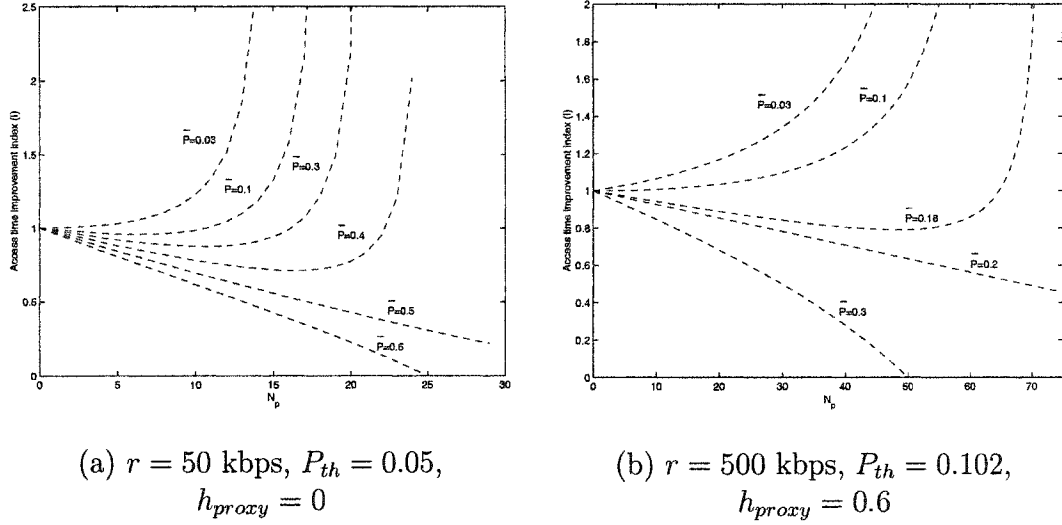


Figure 4.4:  $I$  versus  $N_p$  ( $C = 500$  kbps,  $\lambda = 6.25$  files/s,  $\bar{s} = 40$  kbits,  $N_{on} = 15$  files,  $h_c = 0$ ).

where  $0 \leq K \leq 1 - h_c$ . Based on this relationship, the lowest value  $\Delta_h$  can have is 0 (no prefetching) and the highest value cannot exceed  $1 - h_c$ , since the overall cache hit ratio ( $h_c + \Delta_h$ ) cannot exceed one. Accordingly, the prefetching precision is given by:

$$\bar{P} = \frac{H(1 - e^{-a N_p})}{N_p} \quad (4.19)$$

where  $H \triangleq N_{on} K$ . Because  $\bar{P} \leq 1$ , the constant  $a$  is bounded ( $0 \leq a \leq -\ln(1 - \frac{1}{H})$ ).

Suppose that the prefetcher retrieves files according to their access probabilities, starting from the most likely file. Then increasing  $N_p$  should result in a smaller  $\bar{P}$ , which influences the access time improvement. Consider, for example, the relationship between  $\bar{P}$  and  $N_p$  as defined in (4.19) with  $H = 4.8$  and  $a = 0.16$ . When  $N_p$  changes from 3 to 7,  $\bar{P}$  will change from 0.6 to 0.46. Hence, based on Figure 4.3, when  $N_p = 3$ ,  $I < 1$  (prefetching is beneficial), whereas  $I > 1$  (prefetching is harmful) when  $N_p = 7$ . This example demonstrates that aggressive prefetching may sometimes worsen the performance.

Figure 4.5 shows the performance for the same system shown in Figures 4.3, 4.4-a, and 4.4-b, but with  $\bar{P}$  varying according to (4.19). Consider the case  $R = 1$



and  $h_{proxy} = 0$ . In this case,  $\bar{P} > P_{th}$  whenever  $N_p \leq 7$ . For  $N_p = 7$ , prefetching all seven files with access probabilities greater than  $P_{th}$  improves the performance ( $I < 1$ ), but does not necessarily optimize it (e.g., prefetching 6 files is actually more beneficial than prefetching 7 files). For the other two cases shown in Figure 4.5, we can see that increasing the number of prefetched files can worsen the performance, sometimes even when  $\bar{P} > P_{th}$ .

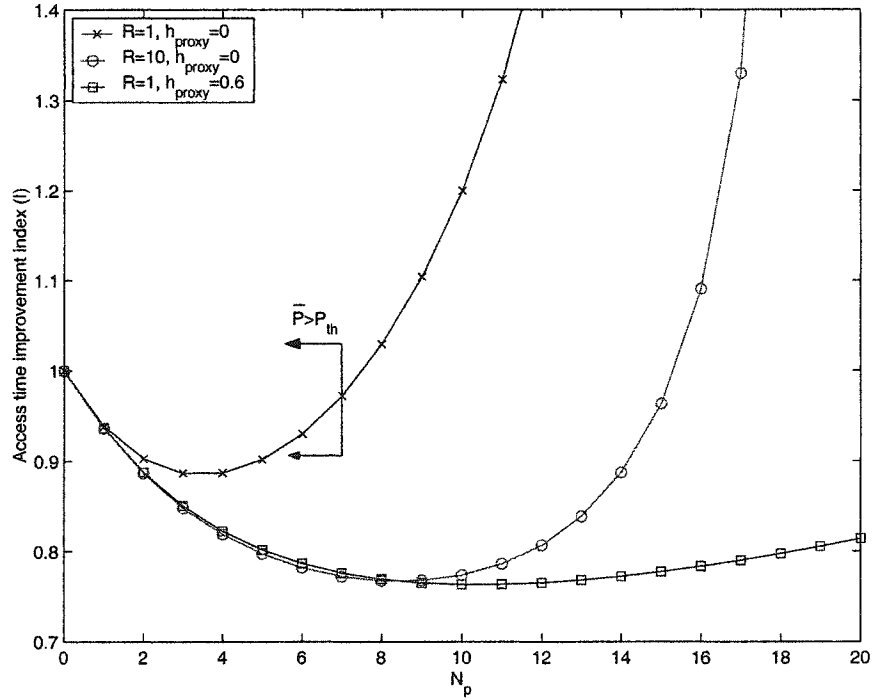


Figure 4.5:  $I$  versus  $N_p$  for the case  $\bar{P} = \frac{4.5(1-e^{-0.24N_p})}{N_p}$  ( $C = 500$  kbps,  $\lambda = 6.25$  files/s,  $\bar{s} = 40$  kbits,  $N_{on} = 15$  files,  $P_{th} = 0.5$ ,  $h_c = 0$ ).

**Corollary 4.4.3** Consider the case when  $R = 1$ ,  $h_{proxy} = 0$ , and  $\bar{P}$  varies with  $N_p$ . Then, prefetching all files with access probabilities greater than  $P_{th}$  reduces the average access time.

**Proof:** The proof follows readily from part 2 of Theorem 4.4.2.

**Theorem 4.4.4** For given  $\bar{P}$  and  $N_p$ , increasing  $R$  or  $h_{proxy}$  reduces the average access time.

**Proof:** See Appendix D.

According to Corollary 4.4.3 and Theorem 4.4.4, a prefetching system can prefetch all files with access probabilities greater than a special threshold value,  $P_{th}$  (the threshold value when  $R = 1$  and  $h_{proxy} = 0$ ). As can be seen in Figure 4.6, this solution reduces the average access time but does not necessarily minimize it with respect to  $N_p$ . This is because for a given  $N_p$ , the worst access delay is when  $R = 1$  and  $h_{proxy} = 0$ .

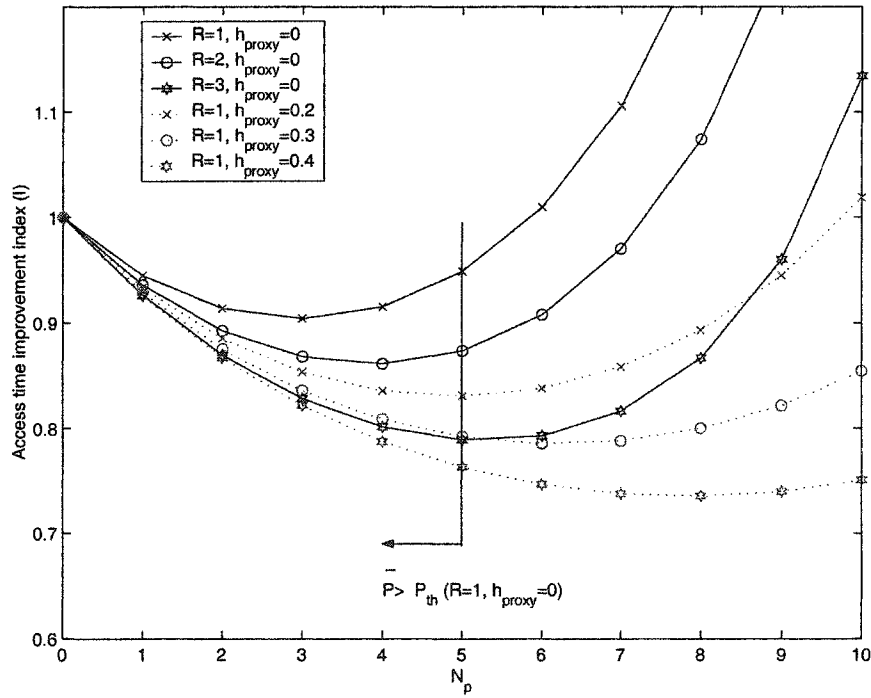


Figure 4.6: Effects of  $R$  and  $h_{proxy}$  on  $I$  for the case  $\bar{P} = \frac{4.5(1-e^{-0.18N_p})}{N_p}$  ( $C = 500$  kbps,  $\lambda = 6.25$  files/s,  $\bar{s} = 40$  kbits,  $N_{on} = 9$  files,  $P_{th}(R = 1, h_{proxy} = 0) = 0.5$ ,  $h_c = 0$ ).

#### 4.5 Prefetching Protocol

In this section, we discuss the applicability of our analytical model in designing a real prefetching system. We first address the issue of estimating the model parameters and then show how such estimates can be used in performing “optimal” prefetching.

The prefetching scheme works as follows. Initially, each client goes through a no-prefetching warm-up period, during which the client estimates its own parameters, including  $h_c$  and  $N_{on}$  (the number of demand-requested files in an ON period). The client also estimates the relationship between  $\bar{P}$  and  $N_p$ . This is done by running the prediction algorithm but without performing any prefetching. Each client reports this information to the proxy, which uses it in estimating  $P_{th}$ . The proxy uses  $P_{th}$  to determine the feasibility of prefetching and to compute  $N_p$  for each client. Moreover, the proxy estimates its own cache hit ratio.

By the end of the warm-up period, the proxy will have computed for each client an approximation of  $h_c$ , the relationship between  $\bar{P}$  and  $N_p$ , and the average length of the ON period. If the proxy determines that prefetching is beneficial (based on  $P_{th}$  and  $\bar{P}$ ), it uses (4.11) to optimize the number of files each client can prefetch. The proxy provides each client with its  $N_p^*$  by piggybacking this information in its response to the client. Once a client has its  $N_p^*$ , it can start prefetching in the subsequent OFF period. We assume that  $N_p^*$  can take non-integer values, where the fractional part means that only a part of a file is prefetched using, for example, the HTTP *range request* [101]. This feature is critical because of the high variability of file sizes in the web. Upon receiving a demand-request, prefetching is stopped and all prefetched data are saved. When a file that was partially prefetched is demand-requested, only the remaining portion of this file is retrieved.

Clients periodically update the proxy with estimates of their parameter values. The proxy uses these estimates along with the estimated load at the proxy ( $\rho_p$ ) to recompute the prefetching parameters ( $P_{th}$  and  $N_p$ ). The recomputation of the prefetching parameters is done based on the variability of the estimated parameters.

Prefetching needs to be implemented fairly for clients with different traffic demands. A reasonable approach is to assign weights to clients depending on their (downlink) traffic demands. The higher the weight assigned to a client, the higher the volume of prefetched traffic that is allowed for that client. The assigned weights can be easily computed by the proxy based on the observed loads of different clients at the shared link. We do not explore this issue further in this paper, and as-

sume a homogenous environment (clients have the same traffic model). Table 4.1 summarizes the main parameters that are used by the prefetching protocol.

Parameter	Definition
$\rho_p$	Average load over the shared link
$h_c$	Hit ratio of the regular cache
$h_{proxy}$	Hit ratio of the proxy cache
$\overline{P}(N_p)$	Prefetching precision as a function of $N_p$
$N_{on}$	Average length of the ON period
$\rho_c$	Client's offered load
$\bar{\rho}_c$	Average client's offered load
$r$	Client access rate (in bps)
$C$	Proxy access rate (in bps)

Table 4.1: Parameters of the prefetching protocol.

#### 4.5.1 Traffic Prediction

Several schemes for WWW traffic prediction have been proposed in the literature (e.g., [16, 65, 57, 45, 94, 34, 79]). Any of these schemes can be integrated into our prefetching protocol. Without loss of generality, we can consider for our simulations the predictor by Jiang et. al [45], with some modifications to include hints from the proxy server. In [45], prediction is done at the client side using the client's history along with hints from the main server. Two types of counters are maintained at the client for each html document: a page counter and a set of link counters. Any time an html document  $X$  is accessed, its page counter  $P_X$  is incremented. If  $X$  has a link to an html document  $Y$  and  $Y$  is accessed from  $X$ , then the link counter  $L_{X,Y}$  is incremented. Following each access to document  $X$ , the predictor computes the probability of accessing every document that is linked from  $X$ . For a linked document  $Y$ , this probability is given by  $\frac{L_{X,Y}}{P_X}$ . If not enough historical information is available for computing this probability, the client relies on hints from the proxy, which runs a similar prediction algorithm but based on the aggregate traffic seen

by all clients. The proxy also maintains some hints from the original servers that can be used if the information collected by the proxy is not statistically sufficient. The prediction algorithm at the proxy requires that clients provide the proxy with information about the html document from which the request is initiated. The proxy also provides the server with similar information.

#### 4.6 Effect of Caching on Prefetching Gain

Intuitively, one may expect that proxy caching has an adverse effect on the prefetching gain. It turns out that this is not always true. For clients with low-bandwidth connections (relative to  $C$ ), the client-proxy link is the bottleneck, and the access time saving due to prefetching a file from the proxy is comparable with the access time saving due to prefetching a file from the original server. As shown in Figure 4.7-a, increasing  $h_{proxy}$  has a negligible effect on the access time saving, especially for a lightly loaded system. At high loads, the gain  $A_{np} - A_p$  increases with  $h_{proxy}$  until a certain point, and after that it stays almost flat. The reason for this increase is that the increase in the load slightly moves the bottleneck to the shared access link, whereas the increase in  $h_{proxy}$  has the opposite effect.

On the other hand, proxy caching is expected to limit the effect of prefetching for clients with high-bandwidth connections and under light load. In this case, the access time saving due to prefetching a file from the original server is considerably higher than the access time saving due to prefetching that file from the proxy server. Figure 4.7-b shows a lightly loaded system (e.g.,  $h_{proxy} = 0.4$  and  $N_p = 4$ ). It can be seen that improving the performance of the proxy cache has a negligible effect on  $A_{np} - A_p$ . On the other hand, when the load is heavy (e.g.,  $h_{proxy} = 0$  and  $N_p = 12$ ), a slight improvement in the performance of the proxy cache can significantly increase the prefetching gain. As the load of the system decreases due to cache performance improvement, this trend is reversed.

The local cache can also limit the number of prefetched files, which in turn limits the prefetching gain. But it also reduces the load, which is advantageous for

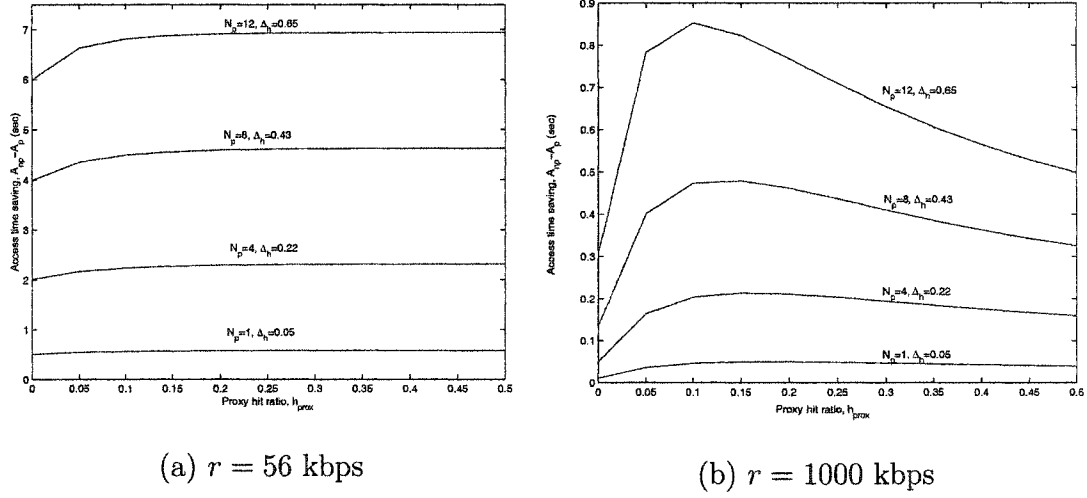


Figure 4.7: Impact of proxy caching on the effectiveness of prefetching ( $C = 1000$  kbps,  $\lambda = 20$  files/s,  $\bar{s} = 40$  kbits,  $N_{on} = 15$  files,  $h_c = 0$ ).

prefetching especially for clients with high-bandwidth connections ( $R = 1$ ) and for a heavily loaded system, as seen in Figure 4.8.

For clients with low-bandwidth connections, local caching has a similar effect to that of proxy caching. The prefetching gain is affected by the local cache only when the system load is high. In this case, increasing  $h_c$  results in an increase in  $A_{np} - A_p$ , as can be seen in Figure 4.9.

In summary, client (and proxy) caching has two opposite effects on the prefetching performance. It has a negative effect by limiting the number of prefetched documents from the original server. But it also has a positive effect by reducing the load on the shared access link. Depending on the confluence of the two effects, the prefetching precision, and the average system load, proxy caching may increase or decrease the effectiveness of prefetching. More importantly, as suggested by the above figures, a certain amount of prefetching is always profitable, irrespective of the client connection speed and the hit rate of the caching system. Moreover, when the system is highly loaded, a modest improvement in the performance of the caching system can significantly increase the prefetching gain.

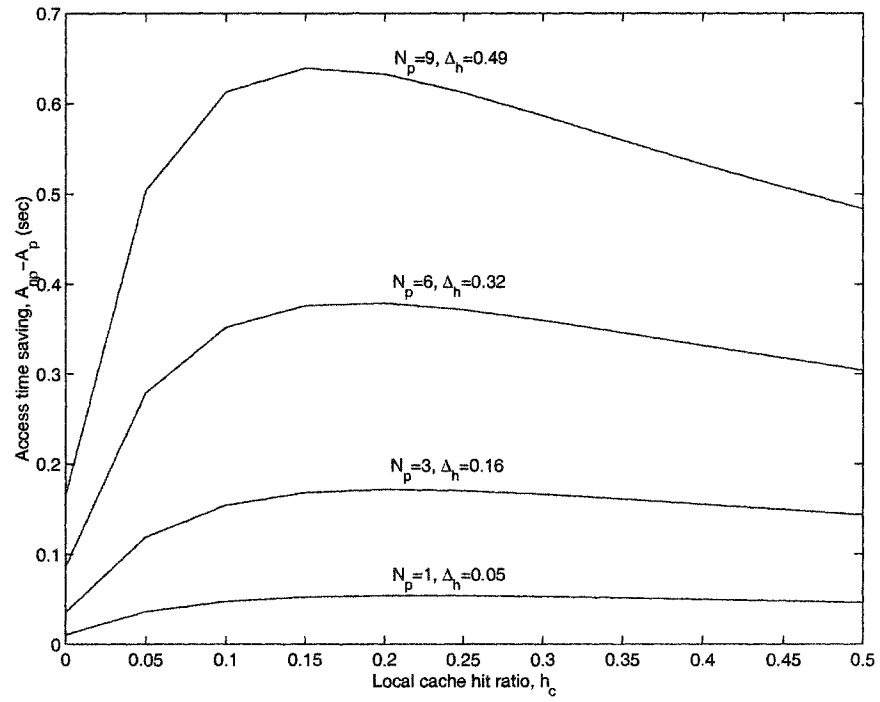


Figure 4.8: Impact of local (regular) caching on the effectiveness of prefetching ( $r = 1000$  kbps,  $C = 1000$  kbps,  $\lambda = 20$  files/s,  $\bar{s} = 40$  kbits,  $N_{on} = 15$  files,  $h_{proxy} = 0$ ).

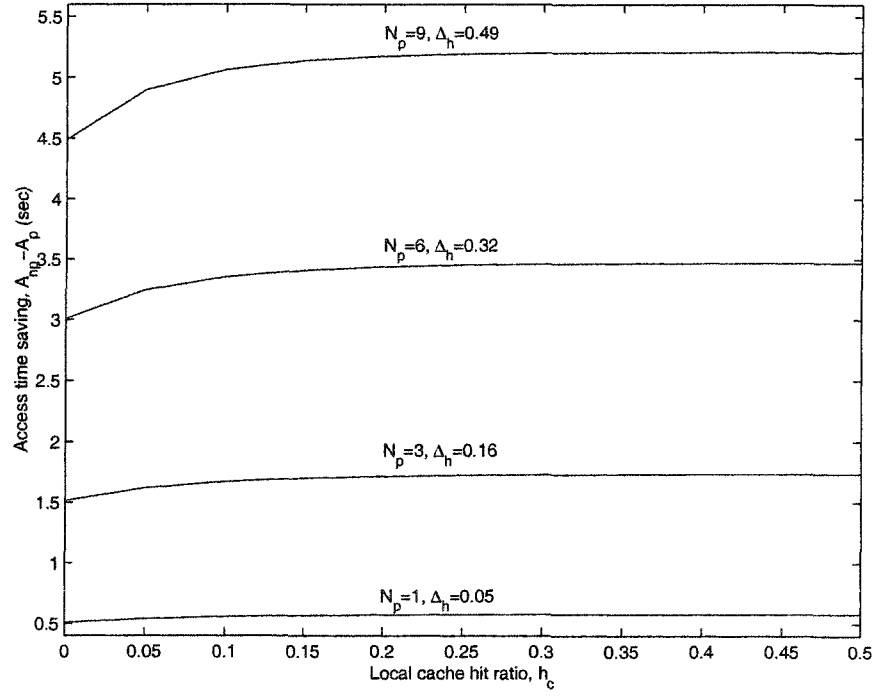


Figure 4.9: Impact of local (regular) caching on the effectiveness of prefetching ( $r = 56$  kbps,  $C = 1000$  kbps,  $\lambda = 20$  files/sec,  $\bar{s} = 40$  Kbits,  $N_{on} = 15$  files,  $h_{proxy} = 0$ ).



## 4.7 Simulation Results

### 4.7.1 Simulation Setup

We consider 50 clients who access the WWW through a common proxy. The proxy cache implements the LRU caching policy with  $h_{proxy} = 0.4$  (the cache hit ratio is controlled by adjusting the cache size). Each client has a large local cache. One percent of this cache is reserved for prefetching. Local caches also implement the LRU caching policy. The shared access link is modeled as an M/G/R-PS queueing system.

### 4.7.2 Traffic Model

We use an ON/OFF source to model the behavior of each client. To mimic the essential properties of web traffic, we rely on an extension of the model in [10] to generate client-side traffic. The model in [10] is based on multifractal processes, which are found to be more flexible than monofractal (self-similar) models in describing traffic “irregularities” at different time scales. This model captures the essential properties of WWW traffic, including temporal locality, spatial locality, and popularity. It allows us to synthesize the aggregate downlink traffic seen by the proxy. This traffic represents responses to requests for main html documents from all clients. Each html document can have one or more inline files (e.g., images). As suggested in [60, 13], the distribution for the number of inline objects in an html document follows a heavy-tail distribution. The OFF period and the file size are also generated according to a heavy-tail distribution [13, 27, 60]. The duration of the ON period is specified by the requested main document and the time it takes the client to retrieve such a document and its inline files. Table 4.2 summarizes the parameters of the most important distributions used in traffic generation.

The model in [10] was not intended for client-side traffic, but rather to capture the properties of the aggregate traffic destined to a group of clients. To synthesize client-side traffic, we start with a no-prefetching simulation run, in which each client is represented by an ON/OFF profile based on the distributions shown in Table 4.2.

Component	Distribution	$f(x)$	Parameters
OFF period	Lognormal	$\frac{1}{x\sqrt{2\pi\sigma^2}}e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$	$\sigma = 1.57$ $\mu = 2.75$
File size	Lognormal	$\frac{1}{x\sqrt{2\pi\sigma^2}}e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$	$\sigma = 1.82$ $\mu = 6.78$
Files per web page	Pareto	$f(x) = ak^ax^{-(a+1)}$	$k = 1$ $a = 1.42$

Table 4.2: Probability distributions used in the simulations.

The aggregate stream is arranged as a vector. When a client starts a new ON period, it selects a document from the top of that vector. This document is considered as the main html document in the current ON period. Each unique document in the vector is assigned a group of unique inline files. Moreover, each file (main document or inline file) is assigned a size that is sampled from the proper distribution. The client retrieves the main document with its inline files from the local cache, proxy cache, or from the original server if the document has not been cached before. The outcome of this simulation run is streams of client requests that are saved in several files to be used in the main simulation experiments. Figure 4.10 depicts an example with three clients. The three clients start their first ON periods at times  $t_1$ ,  $t_2$ , and  $t_3$ , respectively, where  $t_3 > t_1 > t_2$ . According to these times, Client 2 selects the top document ( $A$ ) in the vector of aggregate requests. The first and the third clients select documents  $C$  and  $B$ , respectively. It takes the second client a period of  $dt_1$  seconds to retrieve document  $A$  and its preassigned inline files ( $A_0$  and  $A_1$ ), and it takes it a period of  $dt_2$  seconds to read the retrieved information (OFF period). At the end of the OFF period, this client starts a new ON period, while the other clients are still in their OFF periods. Hence, it selects document  $D$  as the main document, and so on.

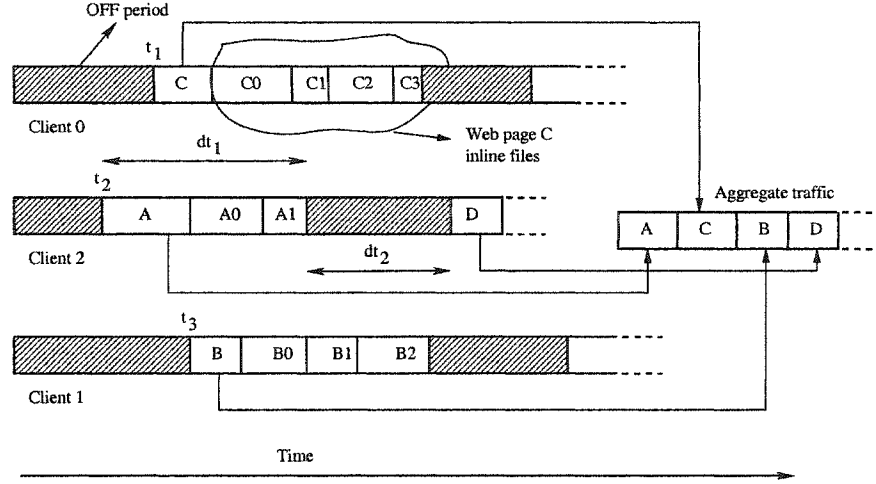


Figure 4.10: Client-side traffic generation process.

#### 4.7.3 Prediction Model

Because prediction is not the focus of our work, for our model-validation purposes, we adopt an artificial predictor whose accuracy can be controlled. The predictor works as follows. Each client is assumed to know the future with certain accuracy and has a window through which it sees this future. To produce a certain relationship between the  $\bar{P}$  and  $N_p$ , the client considers a window of  $m$  files (number of files to prefetch) that are neither in the local cache. Each file is considered for prefetching with probability  $P_i$ , the access probability of the  $i$ th file in the candidate list. If a file is not considered for prefetching, it means that the predictor made a wrong decision. In this case, the client retrieves a dummy file whose size is sampled from the file size distribution. This dummy file is either retrieved from the proxy or the original server based on the estimated value of  $h_{proxy}$ . Figure 4.11 illustrates the main idea behind this artificial predictor. In this figure, the client needs to prefetch three files in the current OFF period. The first three files that are in the future window and are not locally cached are  $B_1$ ,  $B_2$ , and  $C_0$ . To capture a specific relationship between  $\bar{P}$  and  $N_p$ , the access probabilities  $P_1$ ,  $P_2$ , and  $P_3$  for the three candidate files to be prefetched are computed as  $P_i = i\bar{P}(i) - \sum_{j=1}^{i-1} P_j$ ,  $i = 1, 2, 3$ . The client prefetches file  $B_1$  with probability  $P_1$ , and with probability  $1 - P_1$  an alternative dummy file is

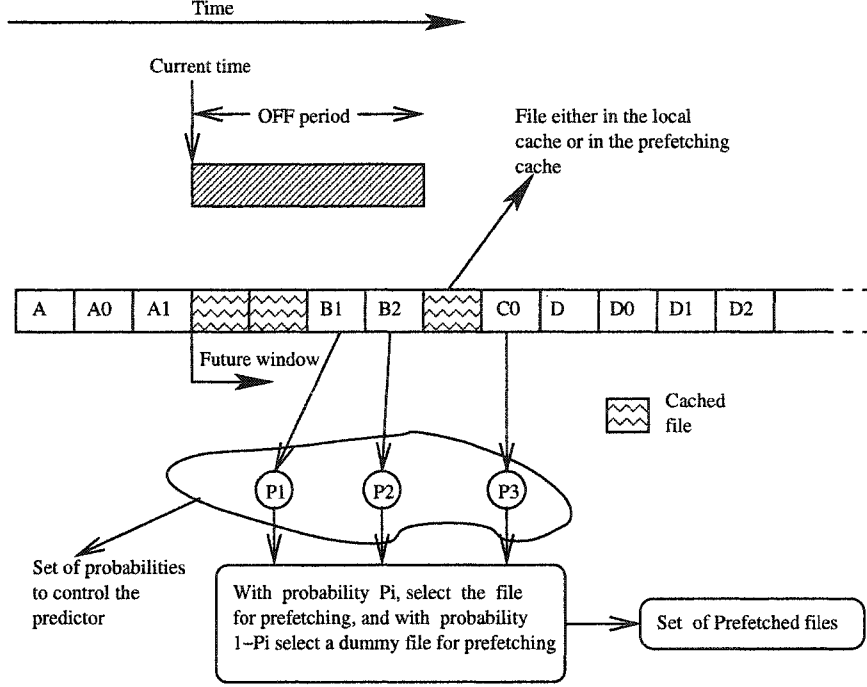


Figure 4.11: Mechanism for traffic prediction.

prefetched. The same thing is done for files  $B_2$  and  $C_0$ . Accordingly, the precision in predicting three files in the future is  $\frac{\sum_{i=1}^3 P_i}{3}$ , which reflects the mimicked  $\bar{P}(N_p)$ .

#### 4.7.4 Validation of $\Delta_h$ and $\rho_p$

In this section, we validate the appropriateness of the models for the increase in the hit ratio due prefetching and the average system load with prefetching. In a given simulation run, each client tries to prefetch a fixed number of files ( $n$ ) in every OFF period, if possible. Each run outputs the access improvement index ( $I$ ), the average hit ratios for all caches, the average system load, and the average number of prefetched documents in an OFF period ( $N_p$ ). Note that  $N_p$  can be less than  $n$  because some OFF periods are not long enough to retrieve all  $n$  files. Figure 4.12 compares the increase in the client cache hit ratio due to prefetching with its numerical counterpart computed using (4.2). It is clear from the figure that the model is very accurate. The average load versus  $N_p$  is depicted in Figure 4.13.

Overall, the modeled and simulated loads are sufficiently close to each other, with a slight deviation when  $N_p$  is high. This deviation comes from the slight change in  $h_{proxy}$  due to prefetching, which we assumed in our analysis to be independent of prefetching. Although we assumed that prefetched documents are not cached in the proxy, prefetching can affect  $h_{proxy}$  as it changes the stream of web requests seen by the proxy.

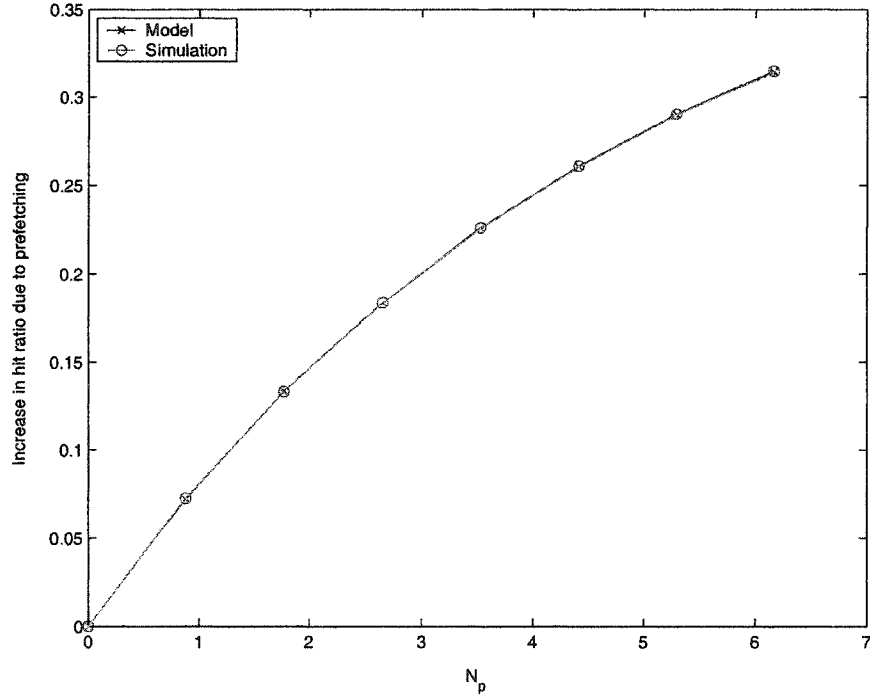


Figure 4.12: Increase in the client's cache hit ratio due to prefetching versus  $N_p$  when  $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$  ( $r = 500$  kbps,  $C = 500$  kbps,  $\lambda = 8$  files/s,  $\bar{s} = 38$  kbits,  $h_{proxy} = 0.39$ ,  $h_c = 0.31$ ).

#### 4.7.5 Validating the Access Improvement Index

Figure 4.14 depicts  $I$  versus  $N_p$ , computed using the analytical model and the simulations. The two plots depict a similar trend. Surprisingly, the prefetching gain in the simulations is lower than the one obtained using the analysis. One reason is related to using the average file size in the analysis, knowing that the file size

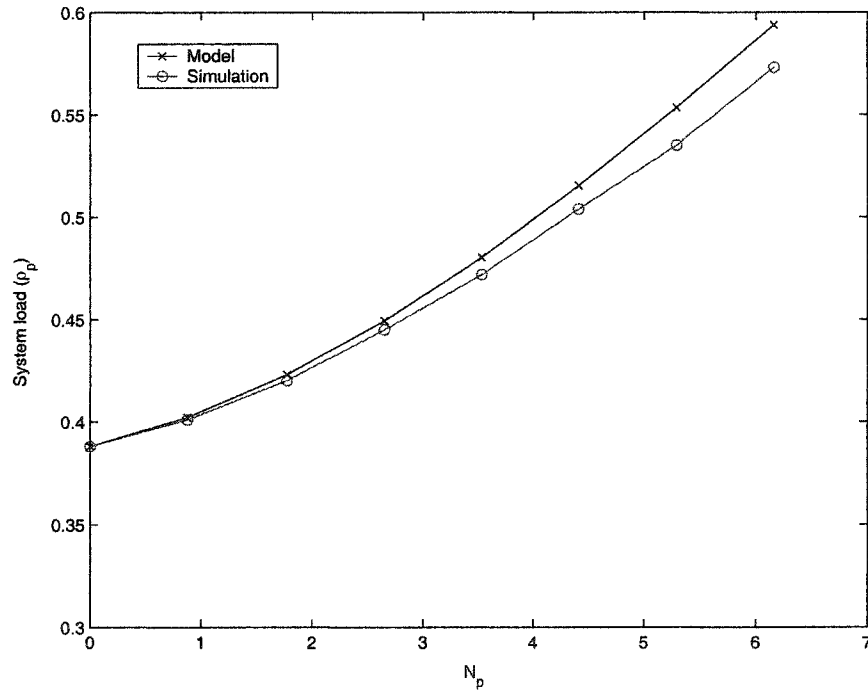


Figure 4.13: Average system load versus  $N_p$  for the case  $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$  ( $r = 500$  kbps,  $C = 500$  kbps,  $\lambda = 8$  files/s,  $\bar{s} = 38$  kbits,  $h_{proxy} = 0.39$ ,  $h_c = 0.31$ ).

follows a heavy-tail distribution (highly variable). To test the effect of the file size on the average access delay, we reran the simulations, assigning to all files the same size (average file size). The outcome of this simulation experiment is shown in Figure 4.15. It is clear that our analysis needs to account for the high variability in the file size. This can be done by modelling the average access delay for a single byte of data. Hence, we use the byte hit ratio of the caching system to compute the probability of finding an arbitrary byte of data in a given cache. Accordingly, the average access time of an arbitrary byte is computed as:

$$A_P(\text{byte}) = \frac{1}{r}(1 - \tilde{h}_c - \widetilde{\Delta}_h)(\widetilde{h_{proxy}} + (1 - \widetilde{h_{proxy}})f_R(\rho_p)) \quad (4.20)$$

where  $\tilde{h}_c$  is the regular-cache *byte* hit ratio,  $\widetilde{h_{proxy}}$  is the proxy cache byte hit ratio, and  $\widetilde{\Delta}_h$  is the increase in the local cache byte hit ratio due to prefetching. To validate this revised model, we reran the simulations to compute the byte hit ratios for all caches. Figure 4.16 shows the numerical results for the original and revised models, along with the simulation results. It is very clear that the results for the revised model are quite close to the simulations.

Based on the revised model, we simulate an adaptive prefetching mechanism, as was described in Section 4.5. Each client dynamically adjusts the number of files to prefetch at the beginning of each OFF period based on the estimated parameters (system load, prefetching precision, and caches *byte* hit ratios). As before, both the regular cache and the proxy cache byte hit ratios are estimated from historical data. The increase in the local cache hit ratio due to prefetching ( $\Delta_h$ ) is estimated based on the number files the client intends to prefetch. The estimated  $\Delta_h$  is used to compute the increase in the local cache byte hit ratio due to prefetching ( $\widetilde{\Delta}_h$ ). This is done by multiplying  $\Delta_h$  by a correction factor  $\alpha$ , which is an estimate of the deviation in the increase in the byte hit ratio to the increase in the file hit ratio due to prefetching. Note that  $\Delta_h$  is estimated for several values of  $N_p$  for optimization purposes ( $N_p$  that gives the best  $I$  is selected). Figure 4.17 shows the simulation results for the adaptive prefetching protocol. In this plot, we also show the results under non-adaptive prefetching, where we run several simulation experiments and

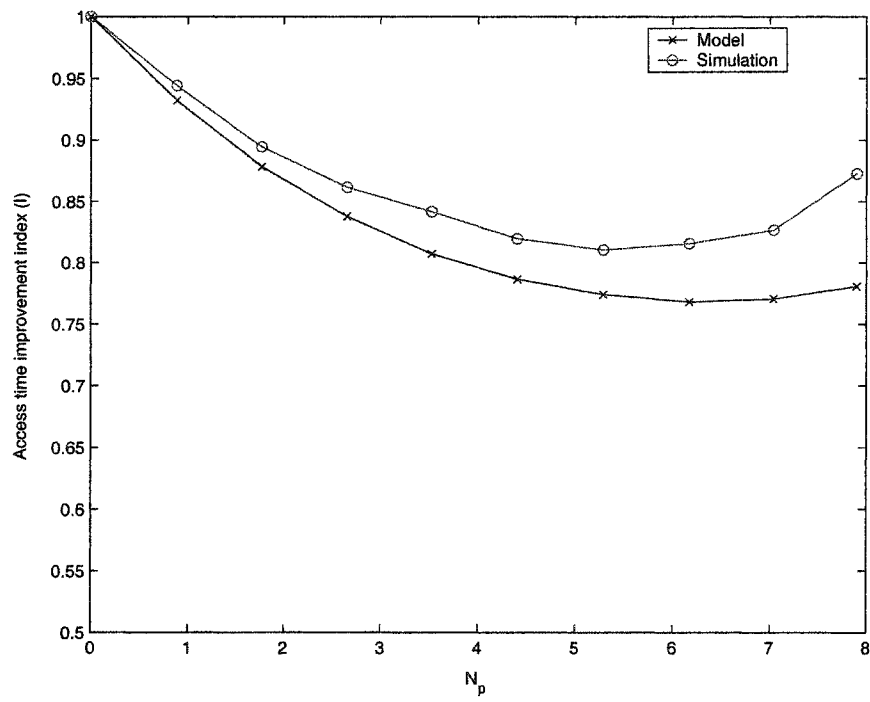


Figure 4.14:  $I$  versus  $N_p$  ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ,  $r = 500$  kbps,  $C = 500$  kbps,  $\lambda = 8$  files/s,  $\bar{s} = 38$  kbits,  $h_{proxy} = 0.39$ ,  $h_c = 0.31$ ).



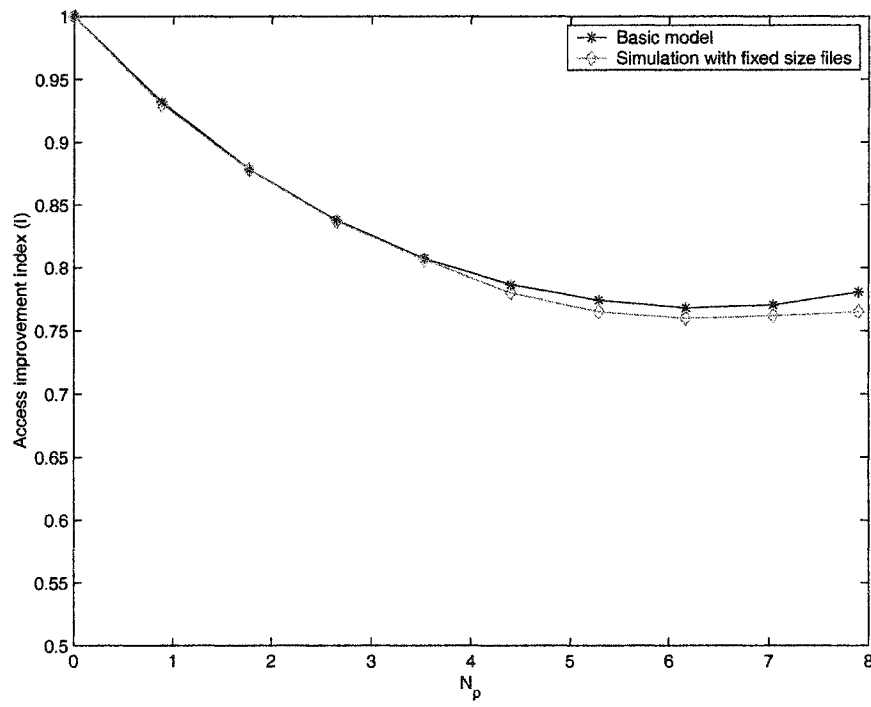


Figure 4.15:  $I$  versus  $N_p$  under fixed-size files ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ,  $r = 500$  kbps,  $C = 500$  kbps,  $\lambda = 8$  files/s,  $\bar{s} = 38$  kbits,  $h_{proxy} = 0.39$ ,  $h_c = 0.31$ ).

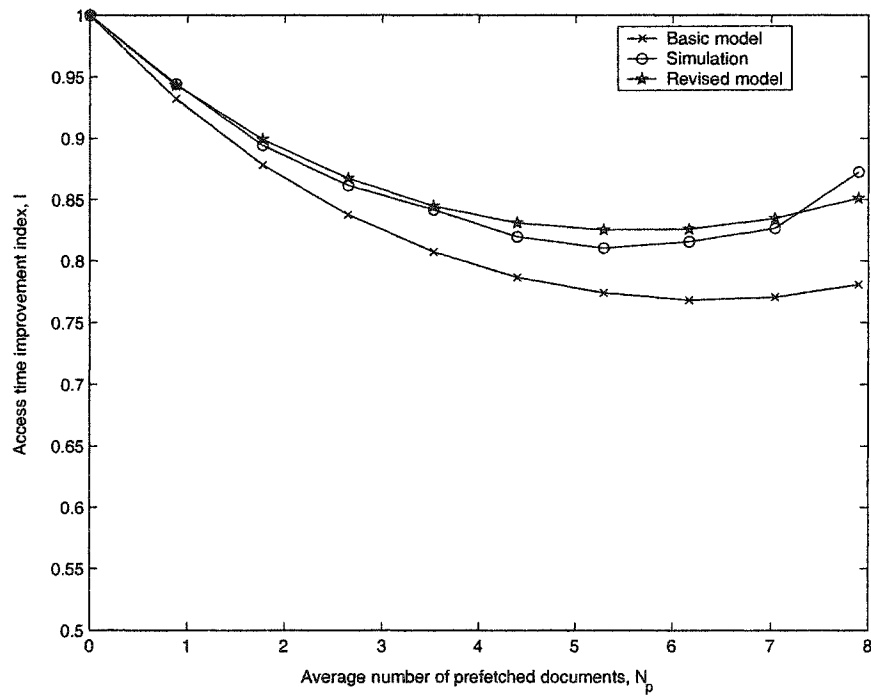


Figure 4.16:  $I$  versus  $N_p$  ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ,  $r = 500$  kbps,  $C = 500$  kbps,  $\lambda = 8$  files/s,  $\bar{s} = 38$  kbits,  $h_{proxy} = 0.39$ ,  $h_c = 0.31$ ).

in each experiment we set  $N_p$  to a given value. From the non-adaptive prefetching simulation, we found that  $N_p^* \approx 5.3$  files. Based on the simulation of the adaptive protocol, the average number of prefetched files was found to be  $N_p = 5.6$ , which is very close to  $N_p^*$ . Moreover,  $I$  for the adaptive protocol is very close to  $I(N_p^*)$ .

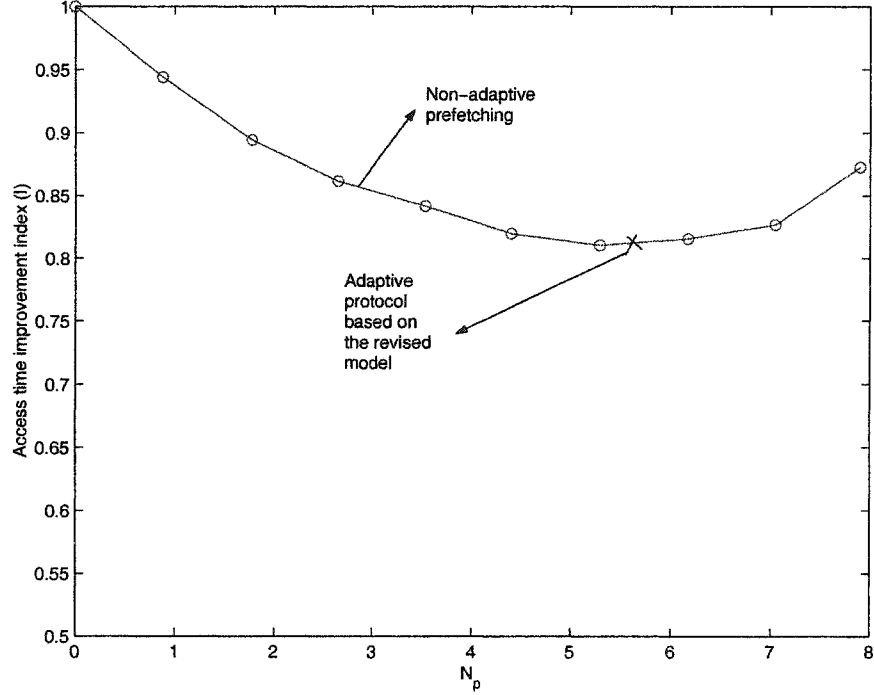


Figure 4.17:  $I$  versus  $N_p$  ( $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ,  $r = 500$  kbps,  $C = 500$  kbps,  $\lambda = 8$  files/s,  $S = 38$  kbits,  $h_{proxy} = 0.39$ ,  $h_c = 0.31$ ).

From the above results, we observe that for higher values of  $N_p$ ,  $I$  in the simulation results tends to slightly deviate from its analytically obtained value. As explained before, this deviation is related to ignoring the change in  $h_{proxy}$  due to prefetching. One more indirect result we discovered from the simulation is that if the number of prefetched files during the OFF periods are highly variable, the degree of burstness in the traffic increases, which can hurt the performance. This is inline with the finding in [24].

## CHAPTER 5

### CONCLUSIONS AND FUTURE WORK

#### 5.1 Conclusions

In this dissertation, we demonstrated the potential of multifractal processes as a viable approach for WWW traffic modeling. We presented two WWW traffic models that use multifractal analysis. While one of the models uses the stack distance approach to capture the main WWW traffic properties, the other model directly captures these properties through the inter-request distance string. As an application for these traffic models, we designed a generic client side prefetching system.

In Chapter 3, we described the multifractal modeling of WWW traffic. We started with the multifractal model of Riedi et al., which is capable of generating approximately lognormal synthetic traces with any desired autocorrelation structure. However, to apply this model in traffic fitting and trace generation, one needs to match as many parameters of the model as the length of the trace to be generated. To make the model parsimonious, we modified it by using a different distribution for the multiplier  $A_j$  (which relates the wavelet and scale coefficients) and by analytically expressing the parameter of  $A_j$ ,  $j = 1, 2, \dots$ , in terms of the mean, variance, and ACF of the modeled data. As a result, the modified multifractal model is specified by five parameters only. We fitted this model to both the normalized stack distance strings and the normalized inter-request strings of different WWW traffic traces. The proposed models capture the spatial and temporal localities of the real traffic as well as the popularity profile. Trace-driven simulations of the LRU cache policy indicate that the proposed models give much more accurate cache miss rates than two previously proposed WWW traffic models. Statistics of the normalized inter-request distances support the goodness of our models. While both approaches, the stack distance approach and the inter-request approach, are superior to previ-

ously proposed models, the accuracy of inter-request distance approach seems better than the stack distance one.

In Chapter 4, we modeled the performance of a generic client side prefetching system. We considered the access time improvement as the performance metric. The model considered both types of caching systems, proxy and client caches. Based on our analysis, we obtained an expression for the prefetching threshold that can be set dynamically to optimize the effectiveness of prefetching. We proposed a prefetching protocol that use the model to optimize the gain out of prefetching. We investigated the effect of the caching system on the effectiveness of prefetching. The main result we discovered is that prefetching all documents that has access probability greater than the optimal threshold value does not always lead to the minimum access delay, as was reported in previous works. This is only true for the case when clients have high access speeds relative to the access speed of their common proxy. For the other cases, the access delay improves with the increase in the number of prefetched documents until a certain point, after which the trend is reversed. Moreover, we found that prefetching is always profitable even with the existence of a good caching system. Another result we found is that the high variability in web file sizes limits the effectiveness of prefetching.

## 5.2 Future Work

The research presented in this dissertation can be extended to different environments such as wireless ad hoc networks. In such environment, each mobile terminal (MT) is equipped with a small storage space that enables the MT to act as a proxy server for a group of neighboring devices. The topic of WWW caching for wireless users is rather new, and not much work has been done in this area. It can be argued, however, that the traffic and prefetching models presented in this dissertation can be adapted for the wireless environment with some modifications to account for the differences in the new environment. For example, prefetching/caching protocols need to account for the MT limitations and the dynamics of the wireless channel.

These limitations include the MT's limited battery life and its small cache size.

For the prefetching model presented in Chapter 4, we assumed that each client can have only one browsing session at a time. The one-session assumption is acceptable for clients of low bandwidth (e.g., dial-up or wireless connections). The case of multiple sessions is more common for clients with high bandwidth connections and multi-user systems, which can be considered for future work.

## APPENDIX A

### PROOF OF THEOREM 3.2.1

The lower bound follows immediately from (3.21) and the fact that  $E[(A^{(2m)})^2] \geq 0$ .

To prove the upper bound, let  $X \triangleq \{X_i : i = 1, 2, \dots\}$  be a positive valued stationary random process, and let  $Y \triangleq \{Y_i : i = 1, 2, \dots\}$  be an aggregation of  $X$  that is defined as follows:

$$Y_n = X_{2n-1} + X_{2n}$$

Note that  $X^{(m)}$  in (3.25) represents  $X$ , while  $X^{(2m)}$  represents  $Y$ . We now prove that  $E[X_n^2]/E[Y_n^2] \leq 0.5$ .

$$\begin{aligned} \frac{E[X_n^2]}{E[Y_n^2]} &= \frac{E[X_n^2]}{E[(X_{2n-1} + X_{2n})^2]} \\ &= \frac{E[X_n^2]}{E[X_{2n-1}^2 + 2X_{2n-1}X_{2n} + X_{2n}^2]} \\ &= \frac{E[X_n^2]}{E[X_{2n-1}^2] + 2E[X_{2n-1}X_{2n}] + E[X_{2n}^2]} \end{aligned}$$

Since  $X$  is stationary, then  $E[X_{2n-1}^2] = E[X_{2n}^2] = E[X_n^2]$ , which leads to:

$$\begin{aligned} \frac{E[X_n^2]}{E[Y_n^2]} &= \frac{E[X_n^2]}{2E[X_n^2] + 2E[X_{2n-1}X_{2n}]} \\ &= \frac{1}{2 + 2\frac{E[X_{2n-1}X_{2n}]}{E[X_n^2]}} \leq 0.5 \end{aligned}$$

since  $E[X_{2n-1}X_{2n}]/E[X_n^2] \geq 0$ .

## APPENDIX B

### PROOF OF THEOREM 4.4.1

First, we show that if prefetching a single file or a fraction of a file does not improve the mean file access time, then increasing the number of prefetched files does not do any better. To do that, we express  $I$  as the product of two functions  $f_1(x)$  and  $f_2(x)$ , where  $x$  is the average number of prefetched files in an OFF period:

$$I = f_1(x) \cdot f_2(x) \tag{B.1}$$

where

$$f_1(x) \triangleq \frac{1 - h_c - \Delta_h(x)}{1 - h_c} \leq 1 \tag{B.2}$$

$$f_2(x) \triangleq A + B f_R(\rho_p(x)) \geq 1 \tag{B.3}$$

$$A \triangleq \frac{h_{proxy}}{h_{proxy} + (1 - h_{proxy}) f_R(\rho_{np})} \tag{B.4}$$

$$B \triangleq \frac{1 - h_{proxy}}{h_{proxy} + (1 - h_{proxy}) f_R(\rho_{np})}. \tag{B.5}$$

We approximate  $f_R(\rho)$  in (4.7) by:

$$f_R(\rho) \approx \frac{1}{1 - \rho^R}. \tag{B.6}$$

The goodness of this approximation is demonstrated in Figure B.1.

It is easy to show that  $f_1(x)$  decreases monotonically with  $x$ , since  $\frac{df_1(x)}{dx} = \frac{-\frac{d\Delta_h}{dx}}{1 - h_c} < 0$  for all  $0 \leq x < \infty$ . Note that  $\frac{d\Delta_h}{dx} > 0$ , as prefetching always increases the overall cache hit ratio.

On the other hand,  $f_2(x)$  increases monotonically with  $x$ , considering that for all  $0 \leq x < \infty$  we have

$$\frac{df_2(x)}{dx} = B \frac{df_R(\rho_p)}{dx} = B \frac{R \rho_p^{R-1}}{(1 - \rho_p^R)^2} \frac{d\rho_p}{dx} > 0. \tag{B.7}$$



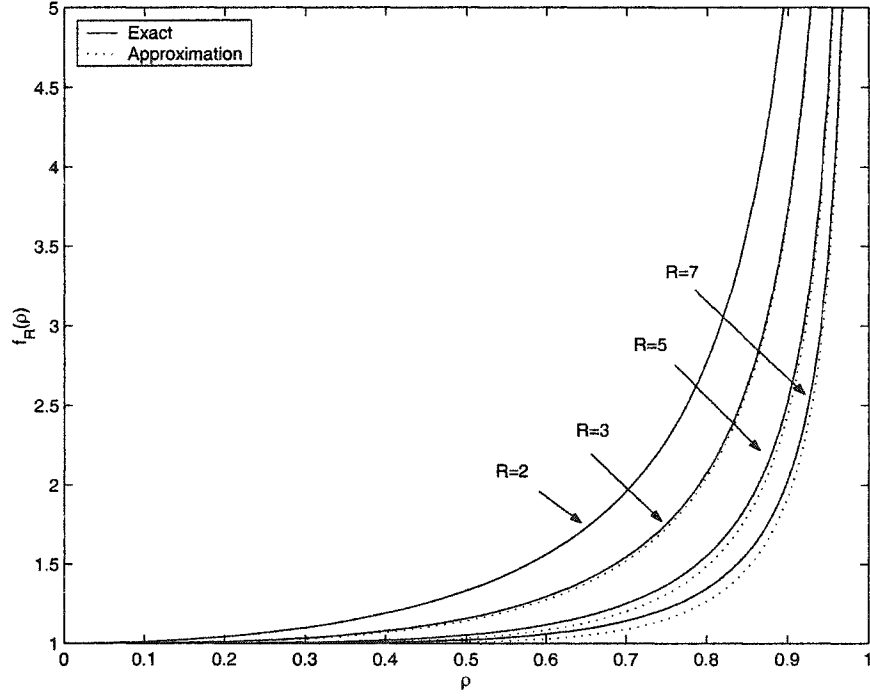


Figure B.1: Approximation of  $f_R(\rho)$  by  $1/(1 - \rho^R)$ .

Note that  $\frac{d\rho_p}{dx} > 0$ , because prefetching always increases the network traffic unless the prediction is 100% accurate.

Now if we can show that  $f_1(x)$  decreases at a slower rate than the rate at which  $f_2(x)$  increases, then we can say for sure that there is no gain out of prefetching more files if prefetching a single or a fraction of a file is not beneficial. This also assures that if there is a gain out of prefetching, then there is a unique value for  $N_p^*$ . Formally, we need to show that  $\frac{d|f_1'(x)|}{dx} < 0$  and  $\frac{d|f_2'(x)|}{dx} > 0$ . Consider the first inequality. Recall that  $\Delta_h = \frac{x \bar{P}(x)}{N_{on}}$ . Then,

$$\begin{aligned} \frac{d|f_1'(x)|}{dx} &= \frac{1}{1 - h_c} \frac{d^2 \Delta_h}{dx^2} \leq 0 \\ &= \frac{1}{(1 - h_c)N_{on}} (\bar{P}''(x)x + 2\bar{P}'(x)) < 0. \end{aligned} \quad (\text{B.8})$$

Note that  $\bar{P}'(x) < 0$  because  $\bar{P}(x)$  is a monotonically decreasing function in  $x$ . Also,  $\bar{P}''(x) < 0$  since the popularity of WWW files follows a Zipf-like distribution

( $P_i \sim \frac{1}{i^\alpha}$ ,  $0 \leq \alpha \leq 1$ ). For  $\frac{d|f_2'(x)|}{dx}$ , we have

$$\frac{d|f_2'(x)|}{dx} = \frac{BR\rho_p^{R-2}((R-1)(1-\rho_p^R) + 2\rho_p^R R(\frac{d\rho_p}{dx})^2)}{(1-\rho_p^R)^2} + \frac{B \cdot R\rho_p^{R-1}}{(1-\rho_p^R)^2} \cdot \frac{d^2\rho_p}{dx^2} > 0. \quad (\text{B.9})$$

Formally, for prefetching to be beneficial, the following condition must be satisfied:

$$\begin{aligned} & \lim_{x \rightarrow 0^+} \left| \frac{df_1}{dx} \right| > \lim_{x \rightarrow 0^+} \left| \frac{df_2}{dx} \right| \\ \text{iff } & \lim_{x \rightarrow 0^+} \left| \frac{\frac{-d\Delta_h}{dx}}{1-h_c} \right| > \lim_{x \rightarrow 0^+} \left| \frac{BR\rho_p^{R-1}}{(1-\rho_p^R)^2} \frac{d\rho_p}{dx} \right| \\ \text{iff } & \lim_{x \rightarrow 0^+} \left| \frac{\frac{-d\Delta_h}{dx}}{1-h_c} \right| > \lim_{x \rightarrow 0^+} \left| \frac{BR\rho_p^{R-1}}{(1-\rho_p^R)^2} (1-h_{proxy}) \left( \frac{1}{N_{on}} - \frac{d\Delta_h}{dx} \right) \frac{\lambda \bar{s}}{C} \right| \\ \text{iff } & \lim_{x \rightarrow 0^+} \left| \frac{\bar{P}'(x)x + \bar{P}(x)}{N_{on}(1-h_c)} \right| > \lim_{x \rightarrow 0^+} \left| \frac{BR\rho_p^{R-1}}{(1-\rho_p^R)^2} (1-h_{proxy}) \left( \frac{1}{N_{on}} - \frac{\bar{P}'(x)x + \bar{P}(x)}{N_{on}} \right) \frac{\lambda \bar{s}}{C} \right| \\ \text{iff } & \frac{\lim_{x \rightarrow 0^+} \bar{P}(x)}{N_{on}(1-h_c)} > \frac{BR\rho_{np}^{R-1}}{(1-\rho_{np}^R)^2} (1-h_{proxy}) \left( \frac{1}{N_{on}} - \frac{\lim_{x \rightarrow 0^+} \bar{P}(x)}{N_{on}} \right) \frac{\lambda \bar{s}}{C}. \end{aligned}$$

Note that  $\lim_{x \rightarrow 0^+} \bar{P}(x) = \bar{P}(1)$ . Defining  $M \triangleq \frac{BR}{(1-\rho_{np}^R)^2} = \frac{(1-h_{proxy})R}{(1-\rho_{np}^R)(1-h_{proxy}\rho_{np}^R)}$ , we end up with

$$\bar{P}(1) > \frac{M\rho_{np}^R}{1 + M\rho_{np}^R}. \quad (\text{B.10})$$

## APPENDIX C

## PROOF OF THEOREM 4.4.2

Let  $x$  be the number of prefetched files. Then,  $I$  can be expressed as

$$I = Lg(x) + \frac{Mg(x)}{1 - \alpha g(x) - \beta x} \quad (\text{C.1})$$

where

$$g(x) \triangleq 1 - h_c - \Delta_h(x) \quad (\text{C.2})$$

$$L \triangleq \frac{h_{\text{proxy}}}{(1 - h_c)(h_{\text{proxy}} + (1 - h_{\text{proxy}})f_R(\rho_{np}))} \quad (\text{C.3})$$

$$M \triangleq \frac{1 - h_{\text{proxy}}}{(1 - h_c)(h_{\text{proxy}} + (1 - h_{\text{proxy}})f_R(\rho_{np}))} \quad (\text{C.4})$$

$$\alpha \triangleq (1 - h_{\text{proxy}})\rho \quad (\text{C.5})$$

$$\beta \triangleq \frac{\alpha}{N_{on}}. \quad (\text{C.6})$$

Now to optimize  $I$ , we let  $\frac{dI}{dx} = 0$  and solve for  $x$ :

$$\frac{dI}{dx} = Lg'(x) + \frac{Mg'(x)(1 - \beta x) + M\beta g(x)}{(1 - \alpha g(x) - \beta x)^2} = 0. \quad (\text{C.7})$$

With  $\Delta_h(x)$  defined according to (4.18),  $g'(x)$  reduces to  $\frac{-\bar{P}}{N_{on}}$ . Solving (C.7) for  $x$  yields

$$x = \begin{cases} \frac{-B - \sqrt{B^2 - 4AC}}{2A}, & \text{if } B^2 - 4AC > 0 \\ \text{Largest number of candidate} & \\ \text{files subject to } \rho_p < 1, & \text{otherwise} \end{cases} \quad (\text{C.8})$$

where,  $A$ ,  $B$ , and  $C$  are given in (4.15), (4.16), and (4.17), respectively.

To prove the second part of Theorem 4.4.2, we know that for prefetching to be

of a value, we must have  $I < 1$ . Therefore,

$$\begin{aligned}
 I &< 1 \\
 \text{iff } \frac{(1 - h_c - \Delta_h)(h_{proxy} + (1 - h_{proxy})f_R(\rho_p))}{(1 - h_c)(h_{proxy} + (1 - h_{proxy})f_R(\rho_{np}))} &< 1 \\
 \text{iff } \frac{(1 - h_c - \Delta_h)}{(1 - h_c)} &< \frac{h_{proxy} + (1 - h_{proxy})f_R(\rho_{np})}{h_{proxy} + (1 - h_{proxy})f_R(\rho_p)} \\
 \text{iff } 1 - \frac{\bar{P}N_p}{(1 - h_c)N_{on}} &< \frac{h_{proxy} + (1 - h_{proxy})f_R(\rho_{np})}{h_{proxy} + (1 - h_{proxy})f_R(\rho_p)}.
 \end{aligned}$$

Taking  $h_{proxy} = 0$ , we end up with

$$\begin{aligned}
 1 - \frac{\bar{P}x}{(1 - h_c)N_{on}} &< \frac{f_R(\rho_{np})}{f_R(\rho_p)} \\
 &< f_R(\rho_{np})(1 - \rho_p).
 \end{aligned}$$

Because  $\rho_p$  is linear in  $x$ , both sides of the above inequality decrease linearly with  $x$ . Hence, if the rate at which the left-hand side (LHS) decreases at  $x = 0$  is greater than the rate of the right-hand side (RHS), then increasing the value of  $x$  increases the reduction in  $I$  (improves  $I$ ).

For the rate of the LHS to be greater than the rate of the RHS, we must have the following:

$$\begin{aligned}
 \frac{\bar{P}}{(1 - h_c)N_{on}} &> \frac{(1 - h_{proxy})(1 - \bar{P})\rho}{(1 - \rho_{np})N_{on}} \\
 \Rightarrow \bar{P} &> \rho_{np}
 \end{aligned}$$

which is the threshold value that is necessary for prefetching when  $R = 1$  and  $h_{proxy} = 0$ .

## APPENDIX D

## PROOF OF THEOREM 4.4.4

Consider  $A_p$  as defined in (4.4) and  $f_R(\rho)$  as defined in (B.6). Then,

$$\frac{dA_p}{dR} = (1 - h_c - \Delta_h)(1 - h_{proxy}) \frac{\rho_p^R \ln(\rho_p)}{(1 - \rho_p^R)^2}, \quad (\text{D.1})$$

which is less than zero because  $\ln(\rho_p) < 0$ . Accordingly, the access time with prefetching decreases with  $R$ .

For  $h_{proxy}$ , we have

$$\frac{dA_p}{dh_{proxy}} = (1 - h_c - \Delta_h) \left( 1 - \frac{1 - (1 - h_{proxy})^R W + R(1 - h_{proxy}^R)W}{(1 - (1 - h_{proxy})^R W)^2} \right) \quad (\text{D.2})$$

where

$$W \triangleq (1 - h_c - \Delta_h + \frac{N_p}{N_{on}}) \frac{\lambda \bar{s}}{C}. \quad (\text{D.3})$$

But  $1 - (1 - h_{proxy})^R W = (1 - \rho_p^R) < 1$ . Therefore,  $1 - (1 - h_{proxy})^R W > (1 - (1 - h_{proxy})^R W)^2$ . Accordingly,  $\frac{dA_p}{dh_{proxy}} < 0$  and the access time with prefetching decreases with  $h_{proxy}$ .

## REFERENCES

- [1] M. Abrams, C. Standbridge, G. Abdulla, S. Williams, and E. Fox. Caching proxies: Limitations and potentials. In *Proceedings of the Fourth International World Wide Web Conference*, Boston University, December 1995.
- [2] C. Aggarwal, J. Wolf, and P. Fellow. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, January/February 1999.
- [3] D. Albrecht, I. Zukerman, and A. Nicholson. Pre-sending documents on the www: A comparative study. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 2, pages 1274–1279, Stockholm, Sweden, July/August 1999.
- [4] V. Almeida, A. Bestavros, M. Crovella, and A. deOliveira. Characterizing reference locality in the WWW. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 92–103, 1996.
- [5] M. Arlitt, R. Friedrich, and T. Jin. Performance evaluation of web proxy cache replacement policies. In *Proceedings of the tenth International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, pages 193–206, 1998.
- [6] M. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a web proxy in a cable modem environment. Technical Report HPL-1999-35R1, Hewlett Packard, Hewlett Packard Labs, 1999.
- [7] M. Arlitt and C. Williamson. Web server workload characterization: The search for invariants. In *Proceedings of the ACM SIGMETRICS Conference*, pages 126–137, 1996.

- [8] M. Arlitt and C. Williamson. Trace-driven simulation of document caching strategies for internet web servers. *Simulation Journal*, 68(1):23–33, 1997.
- [9] H. Bahn, K. Koh, S. H. Noh, and S. L. Min. Efficient replacement of nonuniform objects in web caches. *IEEE Computer*, pages 65–73, June 2002.
- [10] A. Balamash and M. Krunz. WWW traffic modeling: A multifractal approach. *Computer Networks Journal*, 43(2):211–226, October 2003.
- [11] A. Balamash and M. Krunz. An overview of web caching replacement algorithms. *IEEE Communications Surveys and Tutorials*, 6(2):44–56, Second Quarter 2004.
- [12] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns: Characteristics and caching implications. In *Proceedings of the World Wide Web Conference*, pages 15–28, 1999.
- [13] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS Conference*, pages 151–160, 1998.
- [14] T. Bell, J. Cleary, and I. Witten. *Text Compression*. Prentice Hall, 1990.
- [15] A. Bestavros and C. Cunha. Server-initiated document dissemination for the www. *IEEE Data Engineering Bulletin*, 19(3):3–11, September 1996.
- [16] A. Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proceedings of the 4th ACM International Conference on Information and Knowledge Management*, pages 403–410, 1995.
- [17] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the INFOCOM Conference*, pages 126–134, 1999.

- [18] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and System*, pages 193–206, 1997.
- [19] L. Cherkasova and G. Ciardo. Characterizing temporal locality and its impact on web server performance. In *Proceedings of the Ninth International Conference on Computer Communication and Networks (ICCCN)*, pages 434–441, 2000.
- [20] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *Proceedings of the ACM SIGCOMM'98*, pages 241–253, Vancouver, British Columbia, Canada, 1998.
- [21] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *Proceedings of the ACM SIGCOMM'98*, pages 241–253, Vancouver, British Columbia, Canada, August 1998.
- [22] E. Cohen, B. Krishnamurthy, and J. Rexford. Efficient algorithms for predicting requests to web servers. In *Proceedings of IEEE INFOCOM*, volume 1, pages 284–293, New York, March 1999.
- [23] D. Cox. Long-range dependence: A review. *Statistics: An Appraisal*, pages 55–74, 1984. The Iowa State University, Ames, Iowa.
- [24] M. Crovella and P. Barford. The network effects of prefetching. In *Proceedings of IEEE INFOCOM Conference*, pages 1232–1239, 1998.
- [25] M. Crovella and A. Bestavros. Explaining world wide web traffic self-similarity. Technical 1995-015, Boston University, Computer Science Department, 1995.
- [26] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.



- [27] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW client-based traces. *IEEE/ACM Transactions on Networking*, 1(3):134–233, Jan 1999.
- [28] C. Cunha and C. Jaccoud. Determining www user’s next access and its application to pre-fetching. In *Proceedings of ISCC’97: The second IEEE Symposium on Computers and Communications*, pages 6–11, July 1997.
- [29] K. Curewitz, P. Krishnan, and J. Vitter. Practical prefetching via data compression. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 257–266, Washington, D.C., 1993.
- [30] B. Davison and V. Liberatore. Pushing politely: Improving web responsiveness one packet at a time. *Performance Evaluation Review*, 28(2):43–49, September 2000.
- [31] F. Douglass, A. Feldmann, and B. Krishnamurthy. Rate of change and other metrics: a live study of the world wide web. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1997.
- [32] D. Duchamp. Prefetching hyperlinks. In *Proceedings of 2nd USENIX Symposium on Internet Technologies and Systems*, pages 127–138, 1999.
- [33] A. Erramilli, O. Narayan, A. Neidhardt, and I. Sanjeev. Performance impacts of multi-scaling in wide area TCP/IP traffic. In *Proceeding of IEEE INFOCOM’2000*, volume 1, pages 352–359, Tel Aviv , Israel, March 2000.
- [34] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 178–187, May 1999.
- [35] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of ip traffic: a study of the role of variability and the impact of control. In *Proceedings*

- of the *ACM SIGCOMM*, pages 301–313, Cambridge, Massachusetts, August 1999.
- [36] A. Feldmann, A. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of internet wan traffic. In *Proceedings of the ACM SIGCOMM*, pages 25–38, Vancouver, B.C., 1998.
  - [37] A. Feldman, A. Gilbert, W. Willinger, and T. Kurtz. The changing nature of network traffic: Scaling phenomena. In *Proceedings of the ACM SIGCOMM Conference*, pages 5–29, April 1998.
  - [38] A. Foong, Y.-H. Hu, and D. Heisey. Adaptive web caching using logistic regression. In *Proceedings of 1999 IEEE Signal Processing Society Workshop*, pages 515–524, Madison, WI, August 1999.
  - [39] J. Gao and I. Rubin. Multifractal analysis and modeling of long-range dependent traffic. In *Proceedings of the IEEE International Conference on Communications (ICC)*, volume 1, pages 382–386, 1999.
  - [40] A. Gillbert, W. Willinger, and A. Feldmann. Scaling analysis of conservative cascades, with applications to network traffic. *IEEE Transactions on Information Theory - Special Issues of on Multiscale Statistical Signal analysis and its Applications*, 45(3):971–991, 1999.
  - [41] S. Glassman. A caching relay for the World Wide Web. In *Proceedings of the 1st World Wide Web Conference*, pages 69–76, May 1994.
  - [42] C. Huang, M. Devetsikiotis, I. Lambadaris, and A. R. Kays. Modeling and simulation of self-similar variable bit rate compressed video: A unified approach. In *Proceedings of the ACM SIGCOMM Conference*, pages 114–125, 1995.
  - [43] K. ichi Chinen and S. Yamaguchi. An interactive prefetching proxy server for improvement of www latency. In *Proceedings of the Seventh Annual Conference of the Internet Society (INET'97)*, Kuala Lumpur, Malasia, June 1997.

- [44] S. Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, July 2002.
- [45] Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. *IEEE Journal in Selected Areas of Communication*, 17(4):358–368, April 1998.
- [46] S. Jin and A. Bestavros. Popularity-aware greedy-dual size web proxy caching algorithms. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Taiwan, May 2000.
- [47] S. Jin and A. Bestavros. Sources and characteristics of web temporal locality. In *Proceedings of IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Fransisco, CA, August 2000.
- [48] S. Jin and A. Bestavros. Temporal locality in web request streams. In *Proceedings of the ACM SIGMETRICS Conference*, pages 110–111, 2000.
- [49] S. Jin and A. Bestavros. Greedy-dual\* web caching algorithm. *International Journal on Computer Communications*, 24(2):174–183, February 2001.
- [50] R. Klemm. Webcompanion: A friendly client-sideweb prefetching agent. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):577–594, July/August 1999.
- [51] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin. NPS: A non-interfering deployable web prefetching system. In *USENIX Symposium on Internet Technologies and Systems*, 2003.
- [52] T. Kroeger, D. Long, and J. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

- [53] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.
- [54] T. S. Loon and V. Bharghavan. Alleviating the latency and bandwidth problems in WWW browsing. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, pages 219–230, 1997.
- [55] B. Mandelbrot. Intermittent turbulence in self-similar cascades: Divergence of high moments and dimension of carrier. *Journal of Fluid Mechanics*, 62:331–358, 1974.
- [56] P. Mannersalo and I. Norros. Multifractal analysis of real atm traffic: A first look. Technical Report COST257TD, VTT Information Technology, 1997.
- [57] E. Markatos and C. Chronaki. A top-10 approach to prefetching on the web. In *Proceedings of the INET Conference*, 1998.
- [58] R. L. Mattson, J. Gecsei, and I. T. D. Slutz. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.
- [59] J. Mogul. Hinted caching in the web. In *Proceedings of the seventh ACM SIGOPS European workshop*, pages 103–108, Connemara, Ireland, September 1996.
- [60] M. Molina, P. Castelli, and G. Foddis. Web traffic modeling exploiting tcp connections’ temporal clustering through html-reduce. *IEEE Network Magazine*, 14(3):46–55, May 2000.
- [61] M. Nabe and M. Miyahara. Analysis and modeling of world wide web traffic for capacity dimensioning of internet access lines. *Elsevier Performance Evaluation*, 34(4):249–271, December 1998.
- [62] A. Nicholson, I. Zukerman, and D. Albrecht. A decision-theoretic approach for pre-sending information on the www. In *Proceedings of the 5th Pacific Rim*

- International Conference on Topics in Artificial Intelligence*, pages 575–586, Singapore, 1998.
- [63] N. Niclausse, Z. Liu, and P. Nain. A new efficient caching policy for the world wide web. In *Proceedings of the Internet Server Performance Workshop (WISP'98)*, pages 119–128, Madison, WI, USA, June 1998.
  - [64] A. Nogueira, P. Salvador, and R. Valadas. Modeling network traffic with multifractal behavior. *Telecommunication Systems*, 24(2):339–362, December 2003.
  - [65] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. In *Proceedings of the ACM SIGCOMM Conference*, pages 26–36, 1996.
  - [66] T. Palpanas and A. Mendelzon. Web prefetching using partial match prediction. In *Proceedings of Web Caching Workshop*, San Diego, California, March 1999.
  - [67] T. Palpanas. Web prefetching using partial match prediction. Technical Report CSRG-376, University of Toronto, Ontario, Canada, March 1998.
  - [68] A. Pandey, J. Srivastava, and S. Shekhar. Web proxy server with an intelligent prefetcher for dynamic pages using association rules. Technical Report TR-01-004, Department of Computer Science, University of Minnesota, January 2001.
  - [69] V. Paxson and S. Floyd. Wide-area traffic: The failure of wide-area traffic: The failure of poisson modeling. In *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1997.
  - [70] J. Pitkow and P. Pirolli. Mining longest repeated subsequences to predict world wide web surfing. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, USA, October 1999.

- [71] S. Podlipnig and L. Bszrmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398, December 2003.
- [72] K. Psounis and B. Prabhakar. A randomized web-cache replacement scheme. In *Proceedings of the IEEE INFOCOM Conference*, volume 3, pages 1407–1415, April 2001.
- [73] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison Wesley, 1st edition, December 2001.
- [74] R. Riedi, M. Crouse, V. Ribeiro, and R. Baraniuk. A multifractal wavelet model with application to network traffic. *IEEE Transactions on Information Theory*, 45(3):992–1018, April 1999.
- [75] R. Riedi and J. Vehel. Multifractal properties of tcp traffic: a numerical study. Technical Report RR-3129, INRIA Rocquencourt, France, March 1997.
- [76] R. Riedi. Introduction to multifractals. <http://www.dsp.rice.edu/publications/>.
- [77] A. Riedl, T. Bauschert, M. Perske, and A. Probst. Inverstigation of the *M/G/R* processor sharing model for dimensioning IP networks with elastic traffic. In *First Polish-German Teletraffic Symposium PGTSDresden*, September 2000.
- [78] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170, April 2000.
- [79] S. Schechter, M. Krishnanb, and M. Smithc. Using path profiles to predict http requests. In *Proceedings of the 7th International World Wide Web Conference*, pages 457–467, April 1998.
- [80] D. Serpanos, G. Karakostas, and W. Wolf. Effective caching of web objects using zipf’s law. In *Proceedings of IEEE International Conference on Multimedia and Expo*, volume 2, pages 727–730, New York, NY, July/August 2000.

- [81] J. Shim, P. Scheuermann, and R. Vingralek. Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):549–562, July/August 1999.
- [82] D. Starobinski and D. Tse. Probabilistics methods for web caching. *Performance Evaluation*, 46(2-3):125–137, October 2001.
- [83] N. Swaminathan and S. V. Raghavan. Intelligent prefetch in www using client behavior characterization. In *Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 13–19, San Francisco, CA, August 2000.
- [84] N. J. T. Tuah, M. Kumar, and S. Venkatesh. Resource-aware speculative prefetching in wireless networks. In *Wireless Networks*, volume 9, pages 61–72, 2003.
- [85] J. Vehel and R. Riedi. Fractional Brownian motion and data traffic modeling: The other end of the spectrum. *Fractals in Engineering*, pages 185–202, January 1997.
- [86] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, volume 36, pages 329–343, 2002.
- [87] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. The potential costs and benefits of long-term prefetching for content distribution. In *The Sixth Web Caching and Content Distribution Workshop*, 2001.
- [88] Z. Wang and J. Crowcroft. Prefetching in world wide web. In *Proceedings of the Global Internet Symposium*, pages 28–32, 1996.
- [89] S. Williams, M. Abrams, C. Standbridge, G. Abdulla, and E. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM SIGCOMM Conference*, pages 293–305, Stanford University, August 1996.

- [90] W. Willinger, V. Paxson, and M. Taqqu. *Self-similarity and heavy tails: structural modeling of network traffic, a practical guide to heavy-tails: statistical techniques and applications*. Birkhauser, Boston, 1998.
- [91] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Proceedings of the 6th International World Wide Web Conference*, pages 325–334, Santa Clara, CA, April 1997.
- [92] N. Young. Online caching as cache size varies. In *The Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 241–250, 1991.
- [93] G. K. Zipf. Relative frequency as a determinant of phonetic change. Reprinted from the Harvard Studies in Classical Philology, Volume XL, 1929.
- [94] I. Zukerman, D. Albrecht, and A. Nicholson. Predicting users' requests on the WWW. In *Proceedings of the 7th International Conference on User Modeling*, pages 275–284, 1999.
- [95] A study of replacement algorithms for a virtual storage computer. *IBM System*, 5(2):78–101, 1966.
- [96] eAcceleration corporation. Webcelerator help page on prefetching. <http://www.webcelerator.com/webcelerator/prefetch.htm>, 2001.
- [97] . . In *Proceedings of IEEE Global Telecommunications Conference GLOBE-COM*, pages 2518–2522, Inst. of Telecommun., Aveiro Univ., Portugal, November 2002.
- [98] CacheFlow Inc. CacheFlow products web page. <http://www.cacheflow.com/products/>, 2002.
- [99] PeakSoft corporation. PeakJet 2000 web page. <http://www.peaksoft.com/peakjet2.html> , 2002.
- [100] Web 3000 inc. NetSonic internet accelerator web page. <http://www.web3000.com/>, 2002.



[101] <http://www.w3.org/Protocols>.

[102] Internet traffic archive: <http://ita.ee.lbl.gov/>.

[103] RizalSoftware. Net accelerator. <http://www.rizalsoftware.com/>.