NOTE TO USERS

This reproduction is the best copy available.

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TECHNIQUES OF DATA PREFETCHING, REPLICATION, AND CONSISTENCY IN THE INTERNET

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

---

by

Xin Chen

2005

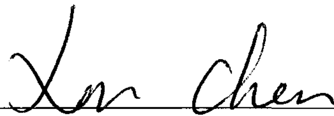
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of

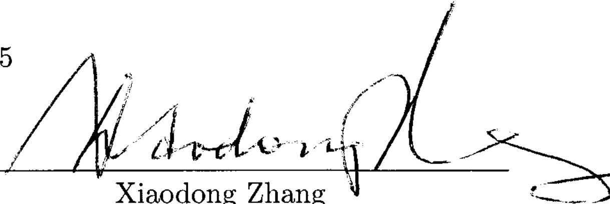
the requirements for the degree of

Doctor of Philosophy

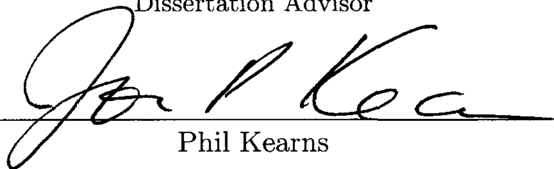


Xin Chen

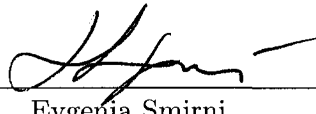
Approved, April 2005



Xiaodong Zhang
Dissertation Advisor



Phil Kearns



Evgenia Smirni

Haining Wang

Zhen Xiao
AT&T Research Lab

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To my parents and dear wife ...

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

Acknowledgments x

List of Tables xi

List of Figures xii

Abstract xv

1 Introduction 2

1.1 Bandwidth vs. Latency 3

1.2 Caching, Replication and Prefetching in Internet 5

1.3 Dissertation Objectives 7

1.4 Dissertation Contributions 8

1.5 Dissertation Organization 10

2 Background of Targeted Internet Techniques 12

2.1 Overview 12

2.2 State of Art of Web Prefetching 12

2.2.1 Types of Web Prefetching 13

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.2 Conditions of Content Prefetching 15

2.2.3 Classifying Prefetching Methods 17

2.2.4 Prefetching Effects 20

2.2.5 Other Variants of Prefetching 22

2.3 DNS Cache Consistency Maintenance 24

2.3.1 DNS Dynamic Update 24

2.3.2 DNS Performance 24

2.3.3 DNS Cache Consistency 26

2.4 P2P Cache Management 28

2.4.1 Cache Management in Unstructured P2P Systems 28

2.4.2 Publish/Subscibe Applications on P2P 29

2.4.3 Wide-Area File Systems on P2P 30

2.4.4 Web Applications on P2P 31

3 Significant Improvement of Web Prefetching **32**

3.1 Introduction 32

3.1.1 Solutions to Reduce Latencies 32

3.1.2 Prefetching in the Web 34

3.2 A Popularity-Based Prediction Model 35

3.2.1 Overview 35

3.2.2 Simulation Environment 36

3.2.3 Surfing Patterns 37

3.2.4 Three Prediction Models 41

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.5 Comparative Performance 46

3.2.6 Summary 51

3.3 Adapting Web Prefetching to Dynamic Server Loads 51

3.3.1 Overview 51

3.3.2 Prefetching Performance Analysis 55

3.3.3 Capacity of A Web Server 57

3.3.4 Average Response Time 59

3.3.5 Adaptive Prefetching Algorithm 61

3.3.6 Prefetching Performance Evaluation 68

3.3.7 Implementation 71

3.3.8 Summary 78

3.4 Coordinated Data Prefetching by Utilizing Reference Information at Both Proxy and Web Servers 79

3.4.1 Overview 79

3.4.2 Evaluation Methodology 80

3.4.3 Limits of Proxy-Based Prefetching 84

3.4.4 Limits of Server-Based Prefetching 86

3.4.5 Coordinated Proxy-Server Prefetching 88

3.4.6 Performance Evaluation 95

3.4.7 Summary 98

3.5 Final Remarks 98

4 DNS Consistency **100**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.1 Introduction 100
4.2 DNS Dynamics Measurement 103
4.2.1 DNS Resource Record Classification 105
4.2.2 Domain Name Collection and Grouping 106
4.2.3 TTLs' Distribution 107
4.2.4 Measurement of Mapping Changes 110
4.3 DNS Cache Update Protocol (*DNScup*) 114
4.3.1 Design Choices 116
4.3.2 Lease Length Effectiveness 118
4.3.3 Dynamic Lease Algorithms 120
4.3.4 Working Procedure of DNScup 123
4.4 Performance Evaluation 125
4.4.1 Poisson Distribution Validation 126
4.4.2 Experimental Results 126
4.5 Prototype Implementation 128
4.5.1 Message Formats 128
4.5.2 Structure of *DNScup* Prototype 129
4.5.3 Secure DNScup 130
4.5.4 Preliminary Results 131
4.6 Summary 133
5 P2P Cache Management in Internet **135**
5.1 Introduction 135

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2 The Base of SCOPE Protocol 138
5.2.1 Overview 139
5.2.2 Partitioning Identifier Space 139
5.2.3 Building Replica-Partition-Trees (RPTs) 141
5.2.4 Load Balancing 144
5.3 Operation Algorithms 145
5.3.1 Subscribe/Unsubscribe 145
5.3.2 Update 149
5.4 Maintenance and Recovery 150
5.4.1 Node Joining/Leaving 150
5.4.2 Node Failure 152
5.5 Performance Evaluation 153
5.5.1 Structure Scalability 154
5.5.2 Operation Effectiveness 155
5.5.3 Maintenance Cost 158
5.5.4 Fault Tolerance 160
5.6 Design Alternatives 161
5.7 Summary 162

6 Conclusion and Future Work **163**
6.1 Existing Problems 163
6.2 Contributions 164
6.3 Future Work 166

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography **168**

Vita **177**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Professor Xiaodong Zhang, for his guidance throughout my years at William and Mary. He is a valuable mentor for both powerful visions and technical depths, and a generous friend for help and care anytime necessary. He has provided us with a first-class research environment where I am growing and becoming mature as a researcher. I am extremely fortunate to have Xiaodong be my advisor in this critical period of my career development.

I would like to thank my other committee members: Phil Kearns, Evgenia Smirni, Haining Wang, and Zhen Xiao for their constructive feedbacks and comments. I give my special thanks to Haining who has made a lot of efforts in my final stage of research. I am also grateful to Professor Bill Bynum for reading many of my papers with critical comments and corrections, and our department administrative director Vanassa Godwin for her advice and help during my PhD study.

The High Performance Computing and Software Lab where I am working has established its tradition and culture of excellence. It has successfully and continuously attracted a large number of talented and hardworking young fellows. I have had many memorable experiences by overlapping with several of them: Songqing Chen, Lei Guo, Song Jiang, Shansi Ren, Li Xiao, Zhao Zhang, and Zhichun Zhu. I have enjoyed tremendously our happiness, inspirations, and friendships.

Finally, I want to express my deepest appreciations to my family. My parents and sister have provided support over all these years. Especially, I would like to thank my wife, Qi for her persistent support and encouragement during my Ph.D. study at the College of William and Mary.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

1.1 Network Bandwidth vs. Latency 4

3.1 Input Parameters for Web Service Models 57

3.2 Characterizations of Different Level (Table Level Demand) 66

3.3 Relationships Among Accuracy, Hit Ratio and Threshold (Table T, A, and P) . . 67

3.4 CPU Utilization Comparison among Different Thresholds 69

3.5 Response Time Comparisons Among Different Thresholds 70

3.6 Merged Boeing Proxy Traces of 5 Days 80

3.7 Selected Scaled-Down Proxy Traces 80

3.8 Selected Pseudo Server Traces 82

3.9 Global Pseudo Server Traces 82

4.1 Measurement Parameters 108

4.2 Average Message Overhead of DNScup 132

6.1 Dissertation Contributions 165

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

1.1 Cache Hierarchy: Computer vs. Internet 6

3.1 Popularity patterns in Web access sessions during day 79 of the WorldCup98 data set 39

3.2 URL popularity and access-session length 41

3.3 Three prediction models 42

3.4 The hit ratio for three PPM models with different thresholds 47

3.5 Traffic overhead comparisons of three models 49

3.6 Space needed measured by nodes 50

3.7 The queuing network model for Web services. 56

3.8 The procedure of computing response time of one device 62

3.9 Request size distributions 65

3.10 The CPU utilization and the service demands of level 2 requests 66

3.11 The request distributions for each level with different thresholds 68

3.12 CPU response time comparisons among different thresholds 70

3.13 The experimental environment 74

3.14 The server throughput and request arrival rates 76

3.15 Server resource utilizations 76

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.16 Server response time 78

3.17 Effectiveness of proxy-based prefetching 85

3.18 Effectiveness of server-based prefetching 87

3.19 Effectiveness of prefetching without coordination 89

3.20 Coordinated and server-based prefetching performance comparisons 91

3.21 The coordinated proxy-server prefetching system design 93

3.22 Hit ratios and byte hit ratios comparisons 95

3.23 Global Web server load comparisons 96

3.24 Global and local network traffic comparisons 97

4.1 The regular domain name distribution with the number of requests in each groups 106

4.2 TTL distributions: (a) All kinds of domain names; (b) .com domain names. 108

4.3 The DN2IP mapping change for each class with different TTLs. 109

4.4 CDN and Dyn domain change frequencies with different TTLs. 112

4.5 The change frequencies of .com domains with different popularity and TTLs. . . . 113

4.6 The resolving latencies for each class with different TTLs. 114

4.7 The space and message changes for fixed length lease schemes. 115

4.8 Example: dynamic lease with different constrains. 117

4.9 DNScup Procedure 122

4.10 DNScup Cache Reference Counter 123

4.11 DNScup Server Lease Threshold 124

4.12 The mean of CV of query interval in DNS traces 124

4.13 Storage requirements of fixed lease and dynamic lease 125

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.14 Query rates of fixed lease and dynamic leas 125

4.15 The Characteristic Fields of a CACHE-UPDATE Message Header 129

4.16 The Structure of DNScup Prototype 129

4.17 DNScup update process. 130

4.18 DNScup Implementation Testbed 131

4.19 DNS nameserver processing overhead: DNScup vs TTL 132

5.1 Identifier space partitioning 139

5.2 Key 5 (101) representative nodes at different levels of partitions 141

5.3 Redundant leaf nodes elimination in RPT 142

5.4 Redundant intermediate nodes elimination in RPT 143

5.5 Node 2 (010) subscribe Key 5 (101) in a 3-bit identifier space. 146

5.6 Level index changes after node 3 joins in a 3-bit space. 146

5.7 Changes of RPT with a node joining 151

5.8 Storage overhead and RPT heights 154

5.9 Number of records kept by each node 156

5.10 Subscribe operation path length comparisons 157

5.11 Distributions of messages sent/received by each node 158

5.12 Maintenance overhead 159

5.13 Effectiveness of update operations with node failure in a 10^4-node network . 159

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ABSTRACT

Internet has become a major infrastructure for information sharing in our daily life, and indispensable to critical and large applications in industry, government, business, and education. Internet bandwidth (or the network speed of transferring data) has been dramatically increased, however, the latency time (or the delay of physically accessing data) has been reduced in a much slower pace. The rich bandwidth and lagging latency can be effectively coped with in Internet systems by three data management techniques: *caching*, *replication*, and *prefetching*. The focus of this dissertation is to address the latency problem in Internet by utilizing the rich bandwidth and large storage capacity for efficiently *prefetching* data to significantly improve the Web content *caching* performance, by proposing and implementing scalable data consistency maintenance methods to handle Internet address *caching* in distributed name systems (DNS), and to handle massive data *replications* in peer-to-peer systems. While the DNS service is critical in Internet, peer-to-peer data sharing is being accepted as an important activity in Internet.

We have made three contributions in developing prefetching techniques. First, we have proposed an efficient data structure for maintaining Web access information, called popularity-based Prediction by Partial Matching (PB-PPM), where data are placed and replaced guided by popularity information of Web accesses, thus only important and useful information is stored. PB-PPM greatly reduces the required storage space, and improves the prediction accuracy. Second, a major weakness in existing Web servers is that prefetching activities are scheduled independently of dynamically changing server workloads. Without a proper control and coordination between the two kinds of activities, prefetching can negatively affect the Web services and degrade the Web access performance. To address this problem, we have developed a queuing model to characterize the interactions. Guided by the model, we have designed a coordination scheme that dynamically adjusts the prefetching aggressiveness in Web Servers. This scheme not only prevents the Web servers from being overloaded, but it can also minimize the average server response time. Finally, we have proposed a scheme that effectively coordinates the sharing of access information for both proxy and Web servers. With the support of this scheme, the accuracy of prefetching decisions is significantly improved.

Regarding data consistency support for Internet caching and data replications, we have conducted three significant studies. First, we have developed a consistency support technique to maintain the data consistency among the replicas in structured P2P networks. Based on Pastry, an existing and popular P2P system, we have implemented this scheme, and show that it can effectively maintain consistency while prevent hot-spot and node-failure problems. Second, we have designed and implemented a DNS cache update protocol, called DNScup, to provide strong consistency for domain/IP mappings. Finally, we have developed a dynamic lease scheme to timely update the replicas in Internet.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TECHNIQUES OF DATA PREFETCHING, REPLICATION, AND CONSISTENCY IN THE INTERNET

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

Data movements between two locations in any part of computer systems are increasingly more critical, expensive, and difficult than computing. Representative examples of the data movement include data flow within CPU via pipeline, between CPU and caches via on-chip and off-chip data links, between caches and DRAM via memory buses, between DRAM and disks via I/O buses, and between a machine to another machine via Internet or wireless connections. Two basic operations are involved for a data movement: (1) transferring data between the two points, for which the data link bandwidth (Bytes/second) determines the transferring speed; (2) the data access time (for physically reading and writing the data), which is called the latency. In the last 20 years, the bandwidths for different types of data links have been dramatically increased, however, the latency time has been reduced in a much slower pace. It is impossible to solve this imbalance problem in computer systems, but the rich bandwidth and lagging latency can be coped in our designs and implementation by three techniques: *caching, replication.* and *prediction* anywhere necessary in computer and distributed systems [96].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The advance of Internet enabled millions of people to access online information and entertainment. For many users, Internet has been a part of their daily life. It has been shown that Internet usage is increased exponentially and this trend appears to be continuing. While advanced networking and computer techniques facilitate the services for clients, the demand on fast response time continues. Even with the rapid increase of network bandwidth, latency remains important for interactive applications across the network. It is critical to keep client-perceived latency low and predictable.

The focus of this dissertation is to address the latency problem in Internet by utilizing the rich bandwidth for accurately *prefetching* data to significantly improve the *Web caching* performance, by proposing and implementing data scalable consistency maintenance methods to handle large scale *data replications* in distributed name systems (DNS) and in peer-to-peer systems.

1.1 Bandwidth vs. Latency

We have observed fast network bandwidth increased for a couple of years and it is expected to increase in a even faster pace in the next decades. The table 1.1 shows the changes of bandwidth in the last ten years.

The network latency has been reduced significantly in recent years, partially due to the increment of network bandwidth. However, the latency reduction does not catch up the bandwidth increase. As Table 1.1 shows, from IEEE standard 802.3 to 803.3ae, the bandwidth is increased 1000 times from 10 Mb to 10000 Mb while the latency is reduced by 94% from 3000 msec to 190 msec. Similar phenomena has been observed in other fields

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in computer science, such as microprocessors, memory and disks.

Table 1.1: Network Bandwidth vs. Latency

LAN	Ethernet	Fast Ethernet	Gigabit Ethernet	10 Gigabit Ethernet
IEEE Standard	802.3	802.3u	802.3ab	802.3ae
Year	1978	1995	1999	2003
Bandwidth (Mb/s)	10	100	1000	10000
Latency (msec)	3000	500	340	190

Three major techniques have been developed to cope with the lagged-latency problem.

- **Caching**: Caches were first attached with processors to overcome long access latency to memory. Caching technique exploits the reference locality by setting a small fast storage to capture most accesses. It has been proved that caching is a simple but very effective way to reduce the latency. Nowadays caches are widely used all over Internet, including all important applications such as DNS, Web, routing, and media streaming.

- **Replication**: Making multiple copies becomes an affordable way to reduce latency, thanks to the increased capacity and decreased price of storage. Most ISPs (Internet Service Providers) use multiple sites spreading across the country to improve the latency to users. Another kind of popular service providers, CDN providers, also relies on replication to improve the quality-of-service for their customers.

- **Prediction**: Prediction aggressively guesses the needs in advance and preloads the data for future use. This technique has been deployed successfully in microprocessor's design and memory management. It is also proposed to be used in Internet to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

improve Web accesses. Related algorithms have been designed to direct the content and location selections for CDN and ISP providers.

1.2 Caching, Replication and Prefetching in Internet

The caching, replication and prefetching techniques have been widely deployed in many fields in computer science. Caches are used to temporarily store commonly used items to provide faster response than that from data sources. For example, modern computer systems take a hierarchical structure to reduce data access latency. High speed memory in microprocessor chips, main memory, local disks and network file systems are coordinated with each other, buffering data at each level for the next. In order to apply caching, replication and prefetching to effectively improve data access latency, two conditions should be satisfied: the cache is faster than the source and the data access patterns are predictable. Data access patterns have been observed in many computer applications, including both space locality and temporal locality.

The three techniques above are particularly attractive for Internet due to high latencies among different hosts. For example, World Wide Web, as one major Internet application, heavily relies on caching for high performance from client-side to server-side. Similarly, we still can regard the cache system as a hierarchical structure, where the browsers work as the caches of the proxies, the proxies as the caches of the servers, and the server local memory/disks as the caches for the server back-ends. Figure 1.1 illustrates the cache structures in both computer systems and Internet. In the last decade, caching, replication and prefetching techniques have been widely deployed in Internet. Nowadays caches are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

important components in Internet infrastructure.

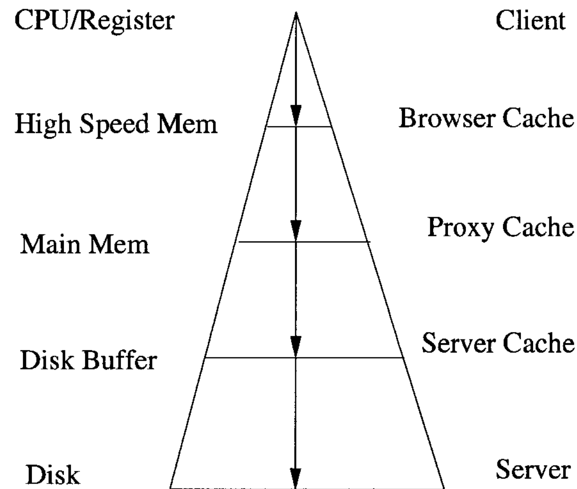


Figure 1.1: Cache Hierarchy: Computer vs. Internet

Different from traditional usages, the deployments of the three techniques meet new challenges in Internet.

- First of all, Internet caching is less effective than computer caching, because of uncacheable objects and poor data locality. For example, around 25% Web responses are uncacheable due to dynamic URLs, uncacheable HTTP methods, and uncacheable HTTP headers. The spatial locality in Web is poor since the number of possible links in one Web page can be large while most clients only visit a small portion of them. The temporal locality in Web is also poor for both browser caches and proxy caches and only 50% Web objects will be accessed again by the same client.

- Second, the data in Internet is dynamically changed. Cache consistency should be considered to ensure the correctness of the replicated data all over the Internet. For example, the durations to change 50% Web objects are 11 days and 4 months for *.com*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and *.gov* domains respectively [37]; 25% new links are created every week and around 80% of the Web links are replaced with new ones after one year [89].

- Third, peer-to-peer structure becomes a promising component co-existing with the traditional client-server model. It is estimated that more than 50% traffic in consumer ISPs is from P2P applications. While caching is critical to improve the performance of P2P systems, caches in P2P should be managed in a distributed way and be able to provide fault-tolerance.

1.3 Dissertation Objectives

The goals of this dissertation are to improve the performance and reliability of Internet by developing methods for data prefetching, Internet cache consistency, new caching schemes in newly-emerged P2P structures. Specifically,

1. We build an efficient popularity-aware data structure for accurate Internet prefetching.
2. We coordinate the service processes and prefetching processes in a Web server to maximize the productivity.
3. We also coordinate the cached information in Web servers and in proxy servers to fully utilize the available prediction hints and the idle bandwidth.
4. We develop an efficient data consistency protocol and its implementation for Domain Name Systems (DNS), a critical caching facility in Internet infrastructure.
5. We develop a scalable data consistency protocol and its implementation for structured P2P systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Our approach is both experimental and modeling oriented. We analyze Internet traces to determine the potential effects of Web prefetching in the Web caching systems. To evaluate the negative effects of caching inconsistency, we conduct extensive measurement on the live Internet. Our proposed methods have been evaluated by both trace-driven simulation and measurement in Internet.

1.4 Dissertation Contributions

This dissertation makes several major contributions in three themes, namely Web prefetching, DNS cache consistency maintenance, and P2P caching management.

Web Prefetching Techniques

- PB-PPM: Popularity-Based Prediction By Partial Matching

  We proposed to build a compact and effective tree-based structure PB-PPM by combining the merits of both access sequences and objects' popularities. Compared with traditional PPM methods, it can largely reduce the space overhead and improve the prediction accuracy.

- Server-Side Adaptive Prefetching Aggressiveness Control

  We developed a new method to adaptively adjust the prefetching activities to the workloads on both server and networks. Based on measuring the server and network utilization, a Web server decides the prefetching aggressiveness and changes the prefetching thresholds dynamically. The adjustment process is performed periodically to minimize the client perceived latency at real time.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Coordinated Web Prefetching Scheme

  We designed a coordinated prefetching scheme to balance the load of Web prefetching on Web servers and proxies. Each of them utilizes its local Web access information to make predictions. In addition to the improvements on prediction accuracy, proxies can reduce the computation overhead on Web servers and the communication overhead.

DNS Cache Consistency

The consistency issue in Web has been investigated in recent years. However, as a major component in Internet, the consistency of DNS caching is ignored improperly. The current TTL-based scheme was designed 20 years ago. With the development of Internet techniques, the usage of DNS has been changed dramatically. Through our extensive measurement experiments, we have observed the mappings between domain names and IP addresses change frequently. In order to make the Internet work efficiently, the inconsistency of DNS caching is not tolerable even for a short period. We proposed DNScup (DNS Cache Update Protocol) to provide a strong cache consistency mechanism besides the traditional TTL-based approach.

P2P Cache Management

Peer-to-Peer has been an inevitable model in distributed computing systems. Although current P2P systems facilitate static file sharing, newly-developed applications demand that P2P systems be able to manage dynamically-changing files. Maintaining consistency between frequently-updated files and their replicas is a fundamental reliability requirement for a P2P system. We presented *SCOPE*, a structured P2P system supporting consistency among a large number of replicas. By building a replica-partition-tree (RPT) for each key,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SCOPE keeps track of the locations of replicas and then propagates update notifications. Our theoretical analyses and experimental results demonstrate that SCOPE can effectively maintain replica consistency while preventing hot-spot and node-failure problems. Its efficiency in maintenance and failure-recovery is particularly attractive to the deployment of large-scale P2P systems.

1.5 Dissertation Organization

The remaining chapters are organized as follows. The next chapter of this thesis surveys work from a number of related fields.

Chapter 3 examines the prefetching methods in details. We first introduce the PB-PPM data structure for prefetching prediction. Then we present the server-side prefetching aggressiveness control based on queuing theory. After that, we consider the coordinations among different Web components (browser, proxy and server).

Chapter 4 investigates the data consistency issues. In this part, we first analyze the tradeoffs between the server-side overhead and the client-perceived latency and deduce adaptive lease algorithm. Then we focus on DNS caching system to validate the importance of strong cache consistency and the effectiveness of adaptive lease. At last, we also discuss the adaptive lease implementations on Bind, the most popular DNS server.

Chapter 5 considers the effective cache management in structured P2P system. Based on DHT, we design a Scalable COnsistency maintenance scheme in structured PEer-to-peer (SCOPE). We first describe the replica-partition-tree (RPT) structure, which is used to manage the cache locations. Then we present SCOPE maintenance operations and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

recovery procedures. We evaluate SCOPE protocol by simulation experiments at the end of this section.

Chapter 6 summarizes our work in Internet latency reduction techniques and discusses their future directions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Background of Targeted Internet Techniques

2.1 Overview

As general solutions for latency reduction, caching, replication and prefetching have been widely deployed in Internet in different places. In this chapter, we will briefly introduce their current developments, especially Web prefetching, DNS cache consistency maintenance, and P2P caching management.

2.2 State of Art of Web Prefetching

A typical Web visit will go through several basic stages: DNS resolution, TCP connection establishment, and HTTP request and response. Prefetching can be applied in advance in all these stages.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.1 Types of Web Prefetching

DNS Prefetching

Before establishing a connection to a Web server, a client must translate the host portion of the requested URL into an IP address. The browser first sends a DNS query to a local name server. The name server will reply directly if it has the answer in its cache. Otherwise, it communicates with other name servers to obtain the address. A name server caches an address for TTL (Time To Live) seconds. The TTL parameter for a particular host is determined by its system administrators, and most TTLs are 24 hours. Waiting for a response from the local DNS server introduces a delay in satisfying the user's request. To avoid this delay, the Web client can initiate the name-to-address translation of the user's request in advance.

A study of the prefetching IP address technique is given in [41, 40]. Prefetching the Web server's IP address reduces user-perceived latency at the risk of imposing additional load on the network and the DNS servers. If a large proportion of Web clients perform DNS prefetching, this can result in a substantial increase in the number of queries handled by DNS servers on the Internet.

TCP Connection Prefetching

Before the Web clients send HTTP requests to the server, TCP connections should be established first. The TCP connection-establishment time includes the round-trip time (RTT) over the network, which is the time it takes for a small packet to travel from the client to the server and then back to the client, retransmissions due to timeouts, and any other processing needed to facilitate the connection. The router is in a *cold state* if it is in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the path of the first IP datagram traversing to a destination. The cold router effect also delays the first RTT of the datagrams. Each router along the way determines the IP address of the next hop and translates the IP address to a wire-level address (MAC address). These operations may require communications with other routers. The results of this processing may be cached by a router for a period of time, to speed up routing of subsequent IP datagrams to the same destination. Therefore, it is conceivable that the first RTT of the datagrams corresponding to the SYN segments will be longer than subsequent ones.

Opening a TCP connection requires a three-way handshake between the client and the server. The latency for this step of Web transfer depends on the round-trip delays between the two hosts, as well as the queuing delay at the server and additional delays to recover from packet loss. To hide these delays from the user, the Web client can establish a TCP connection to the server in advance of the user's request [43, 40].

Content Prefetching

After establishing a TCP connection, a Web client issues a HTTP request to the Web server. The latency in receiving the HTTP response is determined by a variety of factors, including the response generation time at the server, the size of the response message, and the bandwidth available from the server to the client. The client can hide the delay from the user by issuing the HTTP requests in advance and caching the responses.

Users usually browse the Web by following hyperlinks from one Web page to another. Hyperlinks on a page often refer to pages stored in the same server. Typically, there is a pause after each page is loaded, while the user reads the displayed material. This time can be used by the client to prefetch files that are likely to be accessed soon, thereby avoiding retrieval latency if those files are actually requested. The retrieval latency has not actually

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

been reduced; it has just been overlapped with the time the user spends in reading, thereby decreasing the access time.

We will focus on content prefetching methods that have been studied widely.

2.2.2 Conditions of Content Prefetching

In order to make an efficient prefetching to reduce the latency, three conditions are required. First, the documents are possibly prefetched before they are requested. Normally a prediction is made based on an access history. Second, the prefetched documents are fresh when they are requested. Third, there should be a sufficient time interval for prefetching before they are requested.

History Information for Prefetching

Usually, the criteria for deciding whether a Web page should be prefetched can be decided by history information statistically. By calculating the interdependencies of Web page accesses periodically based on the most recent access logs, the server/client can group Web pages with interdependencies higher than a certain threshold for prefetching. The following two predictions can not be made: (1) the initial request, and (2) any sequence of requests that have not been observed before.

Expiration Time

Since Web data can expire before it is demanded, prefetching too far in advance is also a problem. Prefetching can be harmful if prefetched content expires before it is demanded.

A study [56] on a trace gathered at AT&T Labs over several months in 1999 shows that the most common expiration time is zero (60%) of the documents. A zero expiration time does not seem to encourage prefetching because prefetched content will expire once it is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

demanded. However, the meaning of a zero expiration time in practice is "use only once".

The median nonzero expiration time is more than one day, far exceeding the average inter-reference time of 52 seconds. This implies a high-performance potential for prefetching. The median expiration time is large and many pages have not been modified in months.

Time for Prefetching

The thinking time between requests is used by prefetching. Prefetching results can be imperfect if a page is demanded during the time between the prediction of its need and its arrival. In such a case prefetching might still lower the observed latency, but generally such partial success is counted as a prefetching failure.

When one page refers to another, the time available to the prefetching is defined as: the time between (1) the end of the transfer of the requests, and (2) the beginning of the transfer of the referee. Studies [56] show the median inter-reference time between two HTML pages is 52 seconds, while the inter-reference time between a HTML page and an image page is 2.25 seconds. However, a significant fraction of images (32.3%) are requested within one second or less of the referencing HTML page.

By calculating the difference between the estimated end-of-transmission of one request and the start of the next request from the same user, some researchers [61] computed the time for prefetching. Based on the studies on three proxy traces (University of California at Berkeley, Digital Equipment Corporation, and University of Pisa), an inherent idle time between user requests is found. About 40% of the requests are preceded by 2 to 128 seconds of idle time, which is plenty for prefetching.

The results of the two studies are consistent, which means there is plenty time to be used for prefetching. Furthermore, dividing the "total page size" (the size of a HTML page

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

plus all its embedded images) by the inter-reference time, [56] estimates the bandwidth needed to prefetch one page. The median bandwidth is around 5 KByte/s, even within the capacity of a dial-up modem, implying that several pages can be prefetched considering bandwidth constraints.

2.2.3 Classifying Prefetching Methods

Based on the source of the access history used for predictions, we can divide prefetching methods into three categories.

Client-Based Prefetching

The simplest way to do prefetching is based on the client's own access history. Client-based prefetching has gained a lot of attention because of its potential performance benefits without changing Web servers.

As an example of client-based non-greedy prefetching, WebCompanion [75] prefetches documents while the user views the current document. WebCompanion uses the following two principles to make prefetching decisions: (1) If an embedded hyperlink is associated with a document that can be fetched very quickly, there is no need to prefetch it. (2) If the document takes a long time to load, prefetch this document as early as possible. The agent employs a prefetching strategy based on estimated round-trip times for Web resources. It prefetches only selected documents with long retrieval latencies and with a relatively low resource usage during prefetching. WebCompanion also incorporates DNS caching.

Proxy-Based Prefetching

Another way is to make predictions based on the information collected by the proxy. Two types of prefetching are in this category: internal prefetching and external prefetching.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In internal prefetching, there will be no communications between Web servers and proxies.

The proxy, being exposed to the Web accessed by multiple users, can often predict what documents a user might access next. If the documents are cached in the proxy, the proxy can utilize the idle periods to push them to the user, or to have the browser pull them. Since the proxy only initiates prefetches for documents in its cache, there is no extra Internet traffic.

In [61], using traces of modem users' Web accesses, the benefits of each technique and their combinations are investigated. Merely increasing the browser cache size would only reduce client latency by 4%. Combining it with a data compression technique reduces client latency by 14.6%. Perfect prefetching and realistic prefetching combined with a large browser cache and data compression reduce client latency by up to 28.6% and 23.4%, respectively. The results demonstrate prefetching is effective since the modem links to the users often have idle periods between two requests from the same client.

In external prefetching, the proxy will prefetch the possible files to be accessed by clients from the server. Wcol [64] is a proxy software that prefetches documents from Web servers. It parses HTML files and prefetches embedded hyperlinks. Bandwidth usage can be controlled by limiting the number of hyperlinks to be prefetched.

Server-Based Prefetching

Compared with the previous two schemes, in this scheme the information from the server can be more accurate since the Web server has significantly more history information relative to the proxy and the client.

Generally, the server predicts the likelihood that a particular Web page will be accessed next and delivers this information to the client. The contacted client then decides whether

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

or not to prefetch the page. The rationale of this approach is as follows: the server has the opportunity to observe the access histories from several clients and use this information to make predictions; the client is in the best position to decide if it should prefetch files based on its cache status and communication costs. Several possible implementations have been proposed [44, 56, 83, 91] to investigate the potential performance of server-based prefetching schemes. The server-based prefetching methods can be roughly divided into two kinds shown below, PUSH and PULL.

The basic idea of PUSH is that servers (and proxies) publish their most-accessed pages since it is easy for Web servers to generate new Web pages. In the study [83], by using trace-driven simulation with traces from 5 sites, the number of most-accessed pages was investigated. As more pages are prefetched, the percentage of prefetched pages that are eventually used rises quickly and levels off at between 3% and 23%, depending on the trace. The method proposed in [19, 55] is that a server respond to a client's request by sending, in addition to the requested documents, a number of other documents that may be requested by that client in the near future.

The operation of Top-10 prefetching [83] can be conducted in a client-proxy-server framework. Prefetching occurs both at the client and the proxy level. User-level clients prefetch from proxies to match the needs of particular users. Proxies are clients to various popular servers from which they prefetch and cache documents to be served to their own clients. In any case, Top-10 prefetching may be transparent to the user and may cooperate with the caching mechanisms of the browser or the proxy.

In PULL-based solution, servers provide hints for clients (or proxies), and clients decide to prefetch the documents or not. In the work of [91], a server maintains per-client usage

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

statistics and determines possibilities through a graph-based Markov model. When a GET is received, the server calculates a list of its pages that are likely to be requested in the near future, using some probability threshold. This list is appended to the GET response, and the client decides whether or not to actually prefetch. Another HTTP extension allows the client to indicate to the server that a certain GET is a prefetch, so that the server will not recursively compute possibilities for the prefetched page. Trace-driven simulations show that average access time can be reduced by approximately 40%, at the cost of much increased traffic (70%). Another result suggests that prefetching is more beneficial than increasing bandwidth. For example, when prefetching causes a 20% increase in traffic, the latency is lower than it would be without prefetching but demands 20% extra bandwidth.

2.2.4 Prefetching Effects

Latency Reduction Bounds

Prefetching has a significant potential to reduce the latency perceived by the clients. Some researchers explored potential improvements in the WWW that can be achieved by using prefetching.

In the study of [77], total Web access is divided into *internal latencies* in local-area networks and *external latencies* in global-area networks. Based on the studies on Digital Equipment Corporation proxy traces, *external latencies* account for 88% of the total amount of latency. Given options for caching and prefetching, four different methods were examined:

- passive proxy caching with unlimited storage;
- an active cache with local prefetching and unlimited storage;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- server hint-based prefetching alone; and
- an active cache with server hint-based prefetching and unlimited storage.

The upper bounds for each model are set by basing the simulations on some best-case assumptions. Each method works with full knowledge of future events. For passive caching, a previously accessed document that has not been changed is still available from the cache. For local prefetching, since a document must be seen at least once before it can be predicted for prefetching, it is assumed that only the first access to a document will not be prefetched, and that all subsequent accesses will be successfully prefetched. For server hint-based prefetching, prefetching can only begin after the client's first contact with that server. In this first-contact model, upon the first contact from a client, a proxy will simultaneously prefetch all of that client's future requests from that server.

For the workload studied, passive caching, with an unlimited cache storage, can reduce latency by approximately 26%. In contrast, prefetching based on local information gives an approximately 41% reduction in latency. Adding server hints increased this bound to approximately 57%.

Prefetching Effects on Networks

Prefetching, in general, increases the burstiness of individual sources, leading to increased average queue sizes in network switches. However, according to the studies in [48], by controlling the transport rate, applications employing prefetching can significantly improve network performance. By using a simple transport rate control mechanism, a prefetching application can modify its behavior from a distinctly ON/OFF entity to one whose data transfer rate changes less abruptly, while still delivering all the data in a user's

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

actual requests.

Rate-controlled prefetching is based on the observation that when prefetching a document it is not necessary to transfer it at the maximum rate; rather, it is only necessary to transfer it at a rate sufficient to deliver it in advance of the user's request. Thus rate-controlled prefetching lowers the transfer rate during prefetching to a level such that the prefetch is initiated as early as possible, while the last byte of the document is delivered just before the document is requested.

The rate-controlled prefetching scheme is able to reduce the mean queue size significantly. This scheme always improves the performance of prefetching, usually by a factor of at least two, and usually fairly consistently over a range of prefetching hit rates.

Experimental results suggest that even a relatively inaccurate rate-controlled prefetching can be effective to reduce network-induced delays, and reduce the variability of such delays.

2.2.5 Other Variants of Prefetching

Although the principles of prefetching on the Web are similar for all proposed methods, they are different in implementations and applications.

Real-Time Prediction

In most proposed schemes, the predictor tree is widely used to record the history information. However, in [56] only a number of most recent accesses of the current page are used to make real-time predictions. In this scheme, clients send usage reports to servers and the server accumulates the information about same page from all clients. When a client request the page, a usage profile will also be returned to the client and it makes prefetching decisions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Prefetching for Multimedia on the Internet

Prefetching has been proposed to meet the QoS requirements of multimedia data on the Web [71, 74]. Unlike previous techniques, the partial prefetch scheme computes the size of the lead segment optimally, and only a minimum but sufficient amount of data is prefetched and buffered. The remaining segment is fetched if and only if the media is traversed. Thus, it delivers content without any increase in perceived response delay.

Prediction of HTTP Requests for Dynamic Content

Based on access history, it is also possible to predict the HTTP requests for dynamically generated contents. In [109], path profiles are constructed for describing HTTP request behaviors, which are sets of pairs, containing paths and their frequency during the test period. Using path profiles, the requests for dynamic contents can be predicted with a high probability. If they are pre-generated by the server using its CPU idle cycles, the processing time of a significant subset of HTTP transactions can be reduced.

Search Engines

The ability to accurately predict user surfing patterns can lead to improvements in Internet searching applications. For example, Google models [24] a client's random walks over the entire WWW link space. The distribution of the visits is obtained from this model. This distribution is considered for ranking the results of a text-based search engine. Following this reasoning principle, researchers believe that surfing models with a higher predictive accuracy would yield high-quality search engines since the models provide a more realistic view of practical usage.

Recommender Systems

As an important component for personalization solutions, recommender systems [65]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

have become popular, and are commercially used in high-profile Web sites such as `www.amazon.com`. They predict items for clients based on the recommendations or the access history of other clients. Some other tools have become available to suggest related pages to Web surfers. The "What's Related" tool button on the Netscape browser developed in [9], provides recommendations based on possibly related links.

2.3 DNS Cache Consistency Maintenance

2.3.1 DNS Dynamic Update

While DNS caching does not support strong consistency, the DNS Dynamic Update mechanism [103] maintains a strong consistency between the primary master DNS name server of a zone and its slave DNS name servers within the same zone. The DNS Dynamic Update mechanism [103] and its enhanced secure version [119] have been proposed and implemented to support dynamic addition and deletion of DNS resource records within a zone, because of the widespread use of DHCP. According to the DNS Dynamic Update protocol, once the primary master has processed dynamic updates, its slaves will get a new copy of them via zone transfers. People have utilized the DNS Dynamic Update protocol to achieve end-to-end host mobility [112].

2.3.2 DNS Performance

DNS performance at either root DNS servers [26, 52] or local DNS servers and their caching effectiveness [72, 81, 120] have been studied in the past decade. Danzig *et al.* [52] measured the DNS performance at one root server and three domain name servers. They identified

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a number of bugs in DNS implementation, and these bugs and misconfigurations produced the majority of DNS traffic. Brownlee *et al.* [26] gathered and analyzed DNS traffic at the F root server. They found that several bugs identified by Danzig *et al.* still existed in their measurements, and the wide deployment of negative caching would reduce the impact caused by bugs and configuration errors. Observing a large number of abnormal DNS update messages at the top of the DNS hierarchy, Broido *et al.* [25] discovered that most of them are caused by default configurations in Microsoft DHCP/DNS servers. The load distribution, availability and deployment patterns in local and authoritative DNS nameservers have been characterized respectively in [93].

Jung *et al.* [72] measured the DNS performance at local DNS servers (MIT and KAIST) and evaluated the effectiveness of DNS caching. They conducted a detailed analysis of collected DNS traces and measured the client-perceived DNS performance. Based on trace-driven simulations, they found that lowering the TTLs of type `A` record to a few hundred seconds has little adverse effect on cache hit rates; and caching of `NS` records and protecting a single name server from overload are crucial to the scalability of DNS. Instead of collecting data at a few client locations, Liston *et al.* [81] compared the DNS measurements at many different sites, and investigated the degree to which they vary from site to site. They identified the measures that are relatively consistent throughout the study and those that are highly dependent on specific sites. Based on both laboratory tests and live measurements, Wessels *et al.* [120] found that existing DNS cache implementations employ different approaches in query load balancing at the upper levels. They suggested longer TTLs for popular sites to reduce global DNS query load.

Shaikh *et al.* [110] demonstrated that aggressively small TTLs (on the order of seconds)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

are detrimental to DNS performance, resulting in the increases of name resolution latency (by two magnitudes), name server workload and DNS traffic. Their work further confirmed that DNS caching plays an important role in determining client-perceived latency. Wills and Shang [122] found that only 20% of DNS requests are not cached locally and non-cached lookups cost more than one second to resolve. The same authors explored the technique of actively querying DNS caches to infer the relative popularity of Internet applications [121]. Using graphs, Cranor *et al.* [46] identified local domain name servers and authoritative domain name server from large DNS traces, which is useful for locating the related DNS caches.

CoDNS [95] identified internal failures as a major source of delays in the PlanetLab testbed, and proposed a locality and proximity-aware design to resolve the problem. They utilized a cooperative lookup service, in which remote queries are sent out when the local DNS nameserver experiences problems, to mask the failure-induced local delay. In their design, they considered the importance of cache at the local DNS nameserver for providing shared information to all local clients, and avoided a design that makes the cache useless.

2.3.3 DNS Cache Consistency

However, none of the previous work focuses on DNS cache consistency. DNS cache inconsistency may induce a loss of service availability, which is much more serious than performance degradation. By contrast, maintaining strong cache consistency in the Web has been well studied. Liu and Cao showed [82] that achieving strong cache consistency with server invalidation is a feasible approach, and its cost is comparable to that of a heuristic approach like adaptive TTL for maintaining weak consistency. To further reduce the cost of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

server invalidation and its scalability, Yin *et al.* [124] proposed volume lease technique and its extension for maintaining cache consistency [125]. Instead of keeping per-client state, Mikhailov and Wills [85] proposed MONARCH to provide strong cache consistency for Web objects, in which invalidation is driven by client requests. They evaluated MONARCH by using snapshots of collected content.

The adaptive lease algorithm has been proposed in [57] to maintain strong cache consistency for Web contents. A Web server computes the lease duration on-the-fly based mainly on either the state space overhead or the control message overhead. However, in their analytical models, the space and message overhead are considered separately without gauging the possible tradeoffs. Thus, the performance improvement of the adaptive lease algorithm is limited. Cohen and Kaplan [42] proposed proactive caching to refresh stale cached DNS resource records, in order to reduce the name resolution latency. However, the client-driven pre-fetching techniques just reduce the client-perceived latency, but cannot maintain strong cache consistency.

Cox *et al.* [45] considered using the Peer-to-Peer system to replace the hierarchical structure of DNS servers. For example, for a given Web server, we can search a distributed hash table to find its IP address, instead of resolving it by DNS. However, compared with conventional DNS, the main drawback of this alternative approach is the significantly-increased resolving latency, while the approach has a stronger support for fault-tolerance and load-balance.

Based on Distributed Hash Tables (DHTs) [73], Beehive—designed for domain name system [100]—provides O(1) lookup latency. Different from widely used passive caching, it uses proactive replication to significantly reduce the lookup latency. In order to facilitate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Web object references, *Semantic Free Reference* (SFR) [116], which is also based on DHTs [73], has been proposed to resolve the object locations. SFR relies on the caches at different infrastructure levels to improve the resolving latency. Note that these proposed schemes are heavily dependent on the wide deployment of DHTs, and the proposed revolutionary changes to the Internet directory service will take a large amount of time and effort to become a reality.

2.4 P2P Cache Management

As a general solution to improve P2P system scalability and object availability, the replication approach has attracted much attention. However, most proposed replication schemes are focused on how to create replicas. Maintaining consistency among a number of replicas is simply ignored, posing a challenge for building a consistent large-scale P2P system.

2.4.1 Cache Management in Unstructured P2P Systems

Some existing file-sharing P2P systems assume that the shared data are static or read-only, and updates are not addressed in the deployment. Most unstructured P2P systems, including both centralized ones, such as Napster, and decentralized ones, such as Gnutella, do not provide a consistency guarantee among replicas. Researchers have designed algorithms to support consistency in a best-effort way without guaranteeing that every replica is updated. In [53], a hybrid push/pull algorithm is used to propagate updates to related nodes, where flooding is substituted by rumor spreading to reduce communication overhead. At every step of rumor spreading, a node pushes updates to a subset of related nodes it knows, providing uncertain consistency only. Similarly, in Gnutella, Lan *et al.* [79] pro-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

posed using flooding-based active push for static objects and adaptive polling-based passive pull for dynamic objects. However, it is hard to determine the polling frequency and no eventual consistency is provided essentially. In [104], Roussopoulos and Baker proposed an incentive-based algorithm called CUP to cache metadata — lookup results — and keep them updated in a structured P2P system. However, CUP only caches the metadata, not the object itself, along the lookup path with limited consistency support, so that it has no support for maintaining consistency among the replicas of an object.

2.4.2 Publish/Subscibe Applications on P2P

For applications demanding consistency support among replicas, different solutions have been proposed in various P2P systems. Most proposed P2P-based publish/subscribe systems record paths from subscribers to publishers, which are used to propagate the updates from publishers. As an anonymous P2P storage and information retrieval system, FreeNet [38] protects the privacy of both authors and readers. It uses a *content-hash key* to distinguish a file's new version from the old one. The update is routed to other nodes based on key closeness. However, the update is not guaranteed to reach every replica. Based on Pastry, Scribe [28] provides a decentralized event notification mechanism for publishing systems. Each node can be a publisher by creating a topic and any other node can become a subscriber through registration. The paths from subscribers to the publisher are recorded for update notifications. However, the single point of failure problem is inevitable if any node on the path fails.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.4.3 Wide-Area File Systems on P2P

Another kind of major P2P applications are wide-area file systems, where replication is widely used to improve performance. Some proposed systems do not explicitly support consistency, while the others support update propagation by constructing either multicast trees or strong-connected graphs. In [50], a decentralized replication solution is used to achieve practical availability without considering replica consistency. PAST [106] is a P2P-based file system for large-scale persistent storage service. In PAST, a user can specify the number of replicas of a file through central management. Although PAST also utilizes caching to improve client latency, it does not maintain consistency of cached contents. Similarly, CFS [51] is a P2P read-only storage system, which avoids most cache consistency problems by content hashes. Every client itself has to validate the freshness of a received file, and the stale replicas are removed from caches by LRU replacement. OceanStore [78] maintains two-tier replicas: a small durable primary tier and a large, soft-state second tier. The primary tier is organized as a Byzantine inner ring, keeping the most up-to-date data. The replicas in second tier are connected through multicast trees: dissemination trees (d-tree) [35]. Periodic heartbeat messages are utilized for fault resilience, which incurs significant communication overhead. Similar solutions are used in P2P-based real-time multimedia streaming (e.g., Bayeux [128], SplitStream [29]). Pangaea [107] creates replicas aggressively to improve overall performance. By organizing all replicas of a file in a strong-connected graph, an update on one server will be propagated to the others through flooding, thus limit its deployments for small numbers of replicas only. Another work [126] uses automatic replica regeneration to provide higher availability with a small number of replicas,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

which is organized in a lease graph. A two-phase write protocol is applied to optimize reads and linearize the read/write process.

2.4.4 Web Applications on P2P

Most P2P-based Web services rely on time-to-live (TTL) values to refresh the replicas. A P2P-based Web caching can be designed to expand the cache size for higher cache hit ratio. Squirrel [68] is such a system based on the Pastry routing protocol. The freshness of a cached object is determined by the Web cache expiration policy (e.g., TTL field in response headers). In order to facilitate Web object references, *Semantic Free Reference* (SFR) [117], based on distributed hash tables, is proposed to resolve the object locations. SFR relies on the caches at its different infrastructure levels to improve the resolving latency. Similar to existing DNS solutions, it uses TTLs to manage the freshness, which may incur the accessibility problem when the requested objects are moved to new locations. Beehive, designed for domain name system [99, 101], provides O(1) lookup latency. Different from widely used passive caching, it uses proactive replication to significantly reduce the lookup latency. In [62], Gedik *et al.* use a dynamic passive replication scheme to provide reliable service for a P2P Internet monitoring system, where the replication list is maintained by each CQ owner.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

Significant Improvement of Web Prefetching

3.1 Introduction

The World-Wide Web has been widely and heavily used in our daily life since it was invented a decade ago. While advanced networking and computer techniques facilitate the services to the clients all over the world, the demand to Internet performance improvement continues. Fast response time is always demanded as the number of clients and type of clients and more kinds of services are enormously increasing. This chapter will address the Web latency issue by advancing the prefetch techniques.

3.1.1 Solutions to Reduce Latencies

An Internet transaction involves different operations, easily causing high latencies perceived by clients. The affecting factors include network bandwidths, network distances, dynamic

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

traffic patterns, types of services, server's scheduling policies and server's capacity. Many researchers have investigated possible solutions to improve the latency even high speed networks and powerful servers have been quickly deployed. Content Delivery and Distribution Services [1] are proposed to build mirror sites close to the potential clients with the same URLs typed by clients. QoS [59, 111, 21] is used to give priorities to certain clients and to guarantee their specific demands. Different server's scheduling policies [15] are proposed to reduce the average response times. The network traffic influence is also investigated [14, 49, 67]. Web caching [13, 23, 27, 77, 123] has been proposed to reduce the network traffic, server's load and improve the latencies. Due to the existence of an increasingly large number of servers, the chance for a client to revisit the same content in the same server is decreasing [16, 22], which limits the potential benefits of Web caching.

Prefetching is a technique to preload the useful information before the clients really request it, which has been actively used in CPU design, computer memory hierarchy and file systems. Prefetching in the Web is to retrieve the to-be-used data into a cache based on the predictions. As a promising solution to Web access latencies, it obtains the Web data a client is expected to need on the basis of data about that client's past surfing activity. Prefetching adds efficiency because it actively preloads the data for two kinds of clients. For group clients, it preloads commonly shared data objects. For individual clients, it loads each of their popular objects. A study shows that the performance improvement with caching and prefetching can be twice that of caching alone [77]. It has been shown that prefetching is more effective to reduce the latencies than increasing the network bandwidth [91]. This dissertation focuses on the document prefetching in the Internet.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.1.2 Prefetching in the Web

When a client is surfing the Web, there is an idle time between two continuous pages visited by the same client for reading and thinking. The ON-OFF pattern provides rooms of prefetching. In order to conduct successful prefetching, it is desired to predict the next accesses from the clients. Many studies show the accesses from different clients sharing the similar patterns, which means accesses from a single client can be predicted from the others.

Web prefetching has been implemented and tested in a few proxy caches and browsers [5, 75, 64]. However, the concerns of high overhead and low accurate prediction rate prevent this system technique from being widely deployed. There are still several important questions to be answered in order for prefetching to be practically used in the Web:

- How can we accurately predict the future accesses?
- How can we effectively make the predictions by different information components?
- How can we minimize the negative effects of the overhead of prefetching?
- How to implement a prefetching system in Internet?
- How to experimentally evaluate a prefefetching system?

This dissertation aims at addressing these questions. Particularly, in this proposal, we concentrate on the challenges in deployment of prefetching with following projects:

- Improve the prediction accuracy and efficiency by designing and implementing prediction algorithms and data structures.
- Design and implement control mechanisms to adaptively coordinate prefetching activities and dynamic workloads in the Web servers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Design and implement global scheduling schemes to coordinate information usage on both proxy caches and Web servers for effective prefetching.

3.2 A Popularity-Based Prediction Model

3.2.1 Overview

An important prefetching task is to build an effective prediction model and data structure for storing highly selective historical information. The prediction by partial match (PPM) model [39, 70, 108], which is widely used for Web prefetching, makes prefetching decisions by reviewing the URLs that clients on a particular server have accessed over some period. The model structures these URLs in a Markov predictor tree that the server dynamically maintains. (A Markov tree is an m-order context tree that uses m preceding symbols to determine the probability of the next one.)

For a Web server that supports millions of Web pages, however, this kind of prefetching takes too much memory, or storage space. Some variations of the PPM model attempt to avoid this overhead by having the servers collect access information for specified documents in near real time [56], but they sacrifice prediction accuracy because there is less historical information. Recently, making prediction from HTML content is also proposed [54].

We propose a variation of the PPM model that builds common surfing patterns and regularities into the Markov predictor tree. The model assigns long branches to popular URLs – ones that clients access frequently – and shorter branches to less popular URLs. The server dynamically updates the tree with a maximum height for each branch type. Because the root nodes are the most popular URLs – not all URLs, as in the standard PPM model

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

– our model effectively uses the space allocated for storing nodes. It also performs two optimizations: It directly links a root node to duplicated popular nodes in a surfing path to give popular URLs more consideration, and it goes back to the completed tree to remove less popular branches.

Because the tree in our model has varied branch lengths, it effectively negotiates the trade-off between predictive accuracy and memory. By limiting the number of less popular documents (short branches), the tree uses less memory, yet it preserves accuracy because it includes the access information that is most likely to result in a prediction hit.

A study comparing our model's performance with the standard PPM model and the longest repeating sequences (LRS) PPM model [98] demonstrates that our model not only is significantly more space efficient, but also provides the most accurate predictions.

3.2.2 Simulation Environment

To verify our model's memory efficiency, predictive accuracy, and general performance, we conducted trace-driven simulations using the data set from 92 days of requests to the 1998 World Cup site. Our simulation environment consisted of traces, a simulated server in which different PPM models make prefetching decisions, and multiple clients that send requests to the server and receive requested and prefetched data from the server.

The traces, the WorldCup98 data set [8], consisted of all the requests made to the 1998 World Cup Web site between 26 April 1998, and 26 July 1998, which represents 92 days of access. During this time, the site received 1,352,804,107 requests. A 1-second interval is used to record access events.

The simulated server dynamically maintained and updated three PPM models – a stan-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

dard model, the longest repeating sequence PPM model, and our popularity-based model – according to these traces. The server assumed that both proxies and browsers were connected to it. The predictor in the simulator assumes that if an address (or IP) sends more than 1,000 requests per day, it is a proxy; otherwise it is a browser. The predictor assumes that the proxy has a disk cache of 16 Gbytes and that a browser has a cache of 10 Mbytes. Both of them use a standard least recently used cache replacement algorithm.

User identification provides useful information for constructing the prediction structure. Unfortunately, obtaining the HTTP log files that identify users was difficult. Some logs have unique user IDs for clients–for example, HTTP cookies–but this type of log file is not available in the public domain. Thus, we used IP addresses, which may represent proxy servers. We recognize that using IP addresses could introduce some inaccuracy in our simulation, but we do not believe it affects our evaluation of the different prediction models.

Finally, in practice, an HTML document can contain embedded image files. Thus, when a client accessed an HTML file and then accessed an image file within the next 10 seconds, we considered the image file to be embedded in the HTML file. We recorded these embedded files as a part of the HTML files.

3.2.3 Surfing Patterns

We observed several popularity patterns during trace analysis. A URL's popularity is the number of times users access it in a given period. To calculate the popularities for all the URLs in the trace file, we used relative popularity (RP) – the individual URL's popularity divided by the highest popularity in the trace. Thus, if URL A had the highest number of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

accesses, its RP would be 100 percent. If URL B had 10 percent of URL A's accesses, its RP would be 10 percent.

We further ranked URL popularity by four grades:

- grade 3, 10 percent < RP <= 100 percent;
- grade 2, 1 percent < RP <= 10 percent;
- grade 1, 0.1 percent < RP <= 1 percent; and
- grade 0, RP <= 0.1 percent.

We characterized each client's surfing behavior as an access session – the sequence of Web URLs that the client continuously visited. If a client was idle for more than 10 minutes, the next request from that client started a new session.

Each access session is composed of the steps the client takes to complete the session, and each step has a sequence number, starting with step 0, the first URL visited in the session. The total number of steps determines the session length, the number of steps per session varies, and each trace file consists of numerous access sessions.

Thus, a trace file with P sessions is a two-dimensional array RP (i,j), where index i represents the ith session in the trace (i = 1, ..., m), and index j represents the jth step in the session (j = 1, ...). We sorted sessions by length, which decreased as the index increased.

Our trace analysis revealed two important recurring patterns, or regularities. All the evaluation days exhibited these regularities to some degree, but for illustration, we randomly chose results from July 13, the 79th day.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To examine the relationship between URL popularity and access session, we first divided each trace into four session groups, which differed in the popularity grade of their starting URLs (grades 3, 2, 1, and 0, respectively).

The first regularity we observed is that most URLs in a server are not popular files, but most clients start an access session from a popular URL. Figure 3.1 shows these popularity patterns. The percentage of URLs of each grade is illustrated by the outer figure and the percentage of starting URLs in all sessions for each popularity grade is shown in the inner figure. Fewer than one percent of URLs are of grade 3 popularity (10 percent < RP <= 100 percent). In contrast, less popular URLs dominate, with grade 0 URLs represent about 95 percent of the total URLs in that day's traces.

These results offer both good and bad news for building prefetching models. The good news is that paying special attention to popular URLs that are only a small percentage of total URLs can be effective. The bad news is that the accumulated number of accesses to less popular URLs can be large, so focusing on only a small percentage of them in prefetching could result in low hit rates.

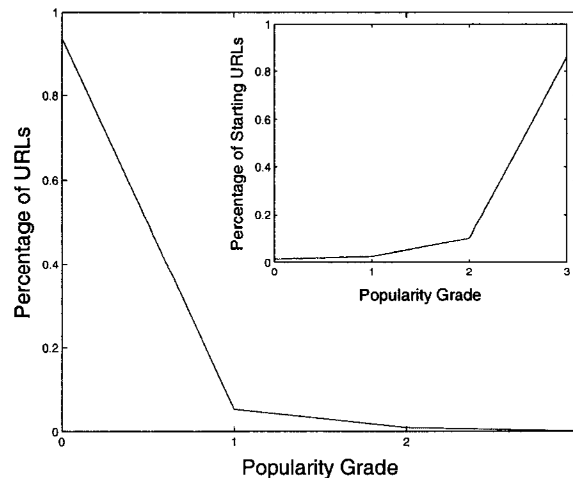


Figure 3.1: Popularity patterns in Web access sessions during day 79 of the WorldCup98 data set

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We then plotted the changing curves using the number of access sessions as a function of session length. Figure 3.2 shows the results for day 79's traces. The outer figure shows the number of sessions remaining as session length increases, and the inner figure demonstrates the average popularity grade across session length. Around 86 percent of the access sessions started from popular URLs, moved to less popular URLs, and exited from the least popular ones. Only 1.3 percent of the sessions started from less popular URLs, remained in the same type of URLs, and exited from the least popular ones.

Thus, the second regularity we observed is that most URL sessions start with a popular URL, move to less popular URLs, and exit from the least popular ones. The least sessions start from less popular URLs, remain in less popular URLs, and exit from the least popular ones.

Our data analysis consistently shows that the starting URLs determined the number of access sessions for a given length. In outer figure in Figure 2, 628,232 sessions started with URLs of grade 3 popularity. As session length increased, the remaining sessions decreased proportionally. For example, 13,293 of the remaining sessions had length 9. In contrast, the remaining sessions with length 9 for sessions starting with URLs of grades 2, 1, and 0 popularity were 4,757, 1,237, and 1,740.

The inner figure in Figure 3.2 shows that the average popularity grade always decreased as the session length increased. For example, the average popularity of the access sessions starting from URLs of grades 3 and 2 popularity decreased proportionally as session length increased. When the session length increased from 0 to 9, the average popularity grade of the URLs decreased from 3 to 1.24 and from 2 to 1.1. For access sessions starting from URLs of grades 1 and 0 popularity, the average popularity changed only slightly as session

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

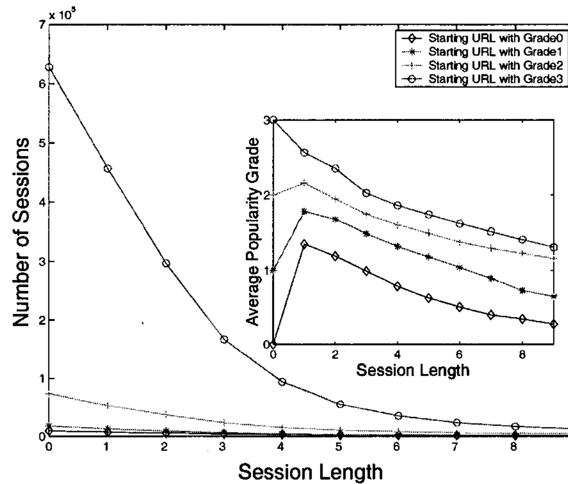


Figure 3.2: URL popularity and access-session length

length increased. Thus, we can infer that clients starting with less popular URLs tend to surf among URLs with the same popularity grade.

3.2.4 Three Prediction Models

To evaluate our popularity-based PPM model against other PPM models, we built three PPM models, shown in Figure 3, into the server as the basis for its prefetching decisions. The server dynamically maintained and updated the PPM models according to traces over the 92-day evaluation period.

Standard model

The first model is the standard PPM model, which left figure in Figure 3.3 shows for three access sequences: $ABCA'B'C'$, ABC, and $A'B'C'$. The standard PPM model uses multiple Markov models to store historical URL paths. Each Markov model partially represents a client session. The model structure is a tree, and each branch is a Markov model with multiple URL predictors. Variable orders in each branch can make predictions.

Node 0 represents the root of the forest. When a client accesses URL A, the server

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

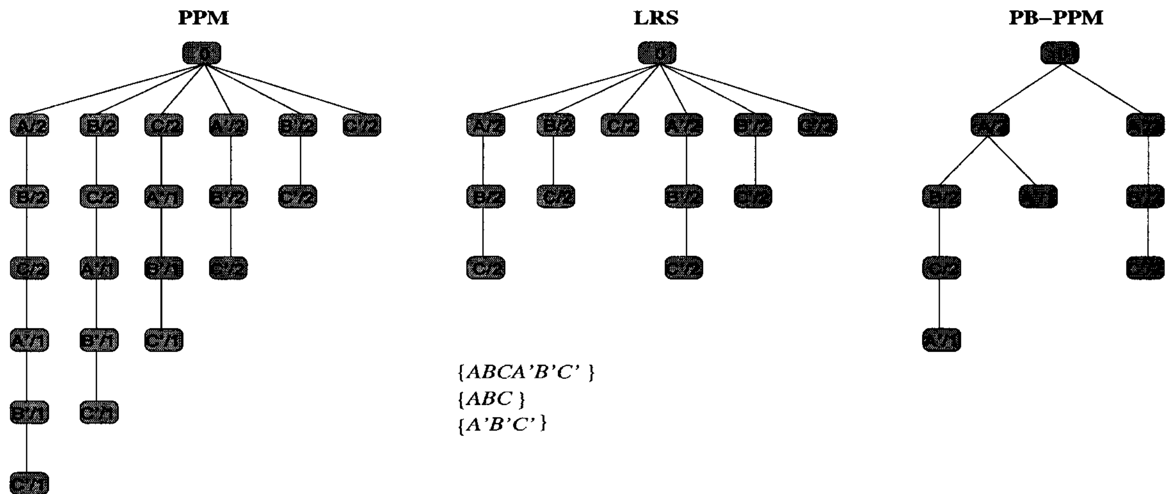


Figure 3.3: Three prediction models for three access sessions– $ABCA'B'C'$, ABC, and $A'B'C'$ – which the server in the trace simulation used to make predictions for prefetching: (left) the standard PPM model, (middle) the LRS PPM model, and (right) the popularity-based-PPM.

builds a new tree with root A and sets the access counter to 1. When B comes, it creates another tree with root B. Because B follows A in the same session, the server must generate another node for B as a child node of A. The process completes until the server has scanned all the URLs accessed in the three sessions. Several prefetching prototypes and systems use this standard model, which follows three main structural rules:

- It uses any URL for a root node and records every subsequent URL in the tree rooted by the first URL.
- Every node represents an accessed URL in the server. A counter records the number of times the URL occurs in the path from the root node. For example, the notation "A/2" indicates that URL A was accessed twice.
- Every path from every root node to a leaf node represents the URL sequence for at least one client.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The advantage of the standard PPM model is that it is not very complex and is thus easy to build. Because the tree records every accessed URL, however, it takes up too much space in the server. Entropy analysis and empirical studies have shown that as the prediction order in each branch increases, so does the space that stores the PPM model's predictors, and prediction accuracy improves.

Some variations of the standard model attempt to fix the tree height (put a ceiling on the number of accessed URLs that can become nodes). This saves storage but the tree no longer matches common surfing patterns. Also, prediction accuracy can be low with a short tree, and even a small height increase can rapidly increase the storage requirement.

For our performance comparison, we used a standard PPM model with a maximum branch height of 7. In practice, the branches in a standard PPM model should have a fixed height, but our experiments show that prediction accuracy will degrade if the branches are too short. We also fixed the height at 7 to make the tree height in the standard model equal to that in our model to provide a more reasonable basis for comparison.

LRS model

The other representative approach to building a PPM model is the LRS PPM model, which keeps the longest repeating subsequences and stores only long branches with frequently accessed URL predictors. A longest sequence is a frequently repeating sequence in which at least one occurrence of one subsequence belongs to an independent access session. Thus, a longest sequence covers many independent access sessions. As in the standard model, the tree height is not fixed. A sequence of URLs that a client accesses more than once is considered a repeated sequence. The middle figure in Figure 3.3 shows the predictor tree structure of the LRS PPM model for the three access sequences. The server builds the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tree the same way as in left figure in Figure 3.3, but it then scans each branch to identify and eliminate non-repeating sequences, such as A'/1, B'/1 and C'/1.

Relative to the standard PPM model, the LRS PPM model offers a lower storage requirement and higher prediction accuracy. Clients access most objects in Web servers infrequently, so keeping only frequently accessed paths does not noticeably affect overall performance, but it does significantly reduce the storage requirement. The high prediction accuracy comes from the model's use of high-order Markov models in a limited number of branches.

The LRS PPM model also has limitations. Because the tree keeps only a small number of frequently accessed branches, it ignores prefetching for many less frequently accessed URLs, so overall prefetching hit rates can be low. Also, to find the longest matching sequence, the server must have all the previous URLs of the current session, which means the server must maintain sessions and update them. This process can be expensive.

For our performance comparison, we used the original LRS design [98]. If clients accessed a URL sequence more than once, we considered it to be frequently repeating.

Popularity-based model

The third approach is our model, which uses only the most popular URLs as root nodes. The right figure in Figure 3.3 shows the tree structure for the three access sequences, where URLs A and A' have grade 3 popularity, URLs B and B' have grade 2 popularity, and URLs C and C' have grade 1 popularity. In this example, the maximum branch height is 4. The server creates a root node only for the starting node and when it detects URLs with grade 3 popularity. Thus, for A it creates a root node, but for B, it creates only a child node of A. It does the same for C. When it detects A' , it generates another root node and a child

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

node of C. It also links child node A' to root node A.

Our popularity-based model builds surfing regularities into the standard PPM model's Markov predictor tree using four rules:

- **Rule 1.** Set the maximum height initially for branches starting from the most popular URLs. The heights of other branches starting from less popular URLs decrease proportionally. Adjust the proportional differences among different branches to adapt to access pattern changes.

- **Rule 2.** Set the initial maximum height by considering the available memory space for the PPM model and access session lengths. The session length reflects the demand for data prefetching. If the lengths of most access sessions are short, building long branches may not be necessary. The maximum height is a moderate number in practice. Our experiments show that more than 95 percent of the access sessions have nine or fewer URLs (or clicks). This is consistent with the results reported elsewhere 9 for a trace file of 3,247,054 Web page requests from 23,692 AOL users on 5 December 1997.

- **Rule 3.** In most cases, add each URL in a sequence only once to the tree. Create a special link between the heading URL and a duplicated node of this URL only if a URL not immediately following the heading URL has a popularity grade higher than the heading URL's grade or has the highest or second highest grade. If this popular node leads to a sequence of URLs, add the sequence to that root or build a new tree rooted by the node. This approach gives popular URLs more considerations for prefetching, aiming at increasing prediction accuracy and access hit ratios.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- **Rule 4.** Periodically build the model on the basis of log files for a previous time interval to predict the surfing patterns in the coming time interval. The interval can be a day, a week, or a month. Dynamically build or adjust the model as each URL request arrives.

Our model also makes space optimizations to the completed tree. The first is based on the relative access probability of nodes (URLs) that are not root nodes, which is the ratio between the number of accesses to a URL and the number of accesses to its parent URL. The server examines each non-root node, and if the node's relative access probability is less than a certain percentage (pre-determined), it removes the node and the branches to its children nodes and all the connected nodes of younger generations. It also removes each node representing a URL that clients accessed only once.

For our performance comparison, we set the maximum branch height to 7 for grade 3 URLs, to 5 for grade 2 URLs, to 3 for grade 1 URLs, and to 1 for grade 0 URLs. We had the server cut each branch with a 5 percent or lower relative access probability.

3.2.5 Comparative Performance

All the PPM models use a longest matching method, which matches as many previous URLs as possible to make a prediction. If the model does not find a prefix match, it will not make a prediction. We set a document selection threshold of 0.1 to 0.9, which meant that the server would prefetch only documents with an access possibility greater than the threshold.

We used three performance metrics:

- The **hit ratio** is the number of requests that hit in a browser or proxy cache as related

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to the total number of requests.

- The **traffic increment** is the ratio between the total number of transferred bytes and the total number of bytes that clients find useful minus 1. The traffic increment is 0 percent if clients find every transferred byte useful.

- The **space** is the required memory allocation measured by the number of nodes available to build a PPM model in the Web server.

The maximum size of prefetched files affects both hit ratios and the traffic increment. A large value lets the server prefetch more data, which helps the hit ratios, but may increase traffic. We set the maximum prefetched file size to 20 KBytes for all three models in all experiments. We also selected day 46 from the WorldCup98 traces, which was one of the busiest days of the evaluation period. We used the day 45 traces as the training data to build the tree structures for the three models.

Hit ratios

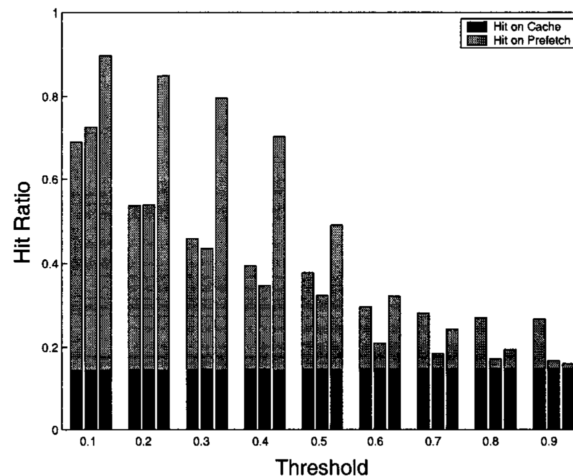


Figure 3.4: The hit ratio for three PPM models with different thresholds

Figure 3.4 shows the changes in the hit ratios using the day 45 traces versus the threshold

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

used for predicting access on day 46. Each set of three bars for each threshold represents hit ratios for the standard PPM model (left bar in each group), the LRS PPM model (middle bar in each group), and our popularity-based PPM model (right bar in each group). The hit ratio is the number of requests that hit in a browser or proxy cache relative to the total number of requests. The thresholds represent the probability that clients will access that document. The threshold 0.3, for example, contains the hit ratios (vertical bars) when the server prefetches documents with an access probability of 0.3.

As the figure shows, the hit ratios were consistently higher with our popularity-based PPM model than with the standard PPM and LRS PPM models. This was generally true of the entire evaluation period (not just day 45). For example, with a 0.3 threshold, the hit ratio of our model is 81 percent – 78 percent higher than the LRS PPM or the standard PPM model. With a larger threshold, however, the hit ratio of our model was the lowest because at the higher thresholds, unpopular files (files that clients have accessed only once) become dominant and available for prefetching. Thus, the standard PPM model achieves almost the same hit ratio at thresholds of 0.6 to 0.9. The access probability of unpopular files is 1.0 when clients access two unpopular files continuously.

Both the LRS PPM model and our model use space optimization to keep popular files, which means that the server will never prefetch unpopular files. However, our model's hit ratio decreases faster than the LRS PPM model's hit ratio because the LRS PPM model uses all URLs as roots, whereas our model uses only popular URLs as root nodes.

Because our model deletes URLs that clients seldom access, it has more flexibility than the other models. This is important when the server load and network conditions dynamically determine how aggressive prefetching can be.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Traffic overhead

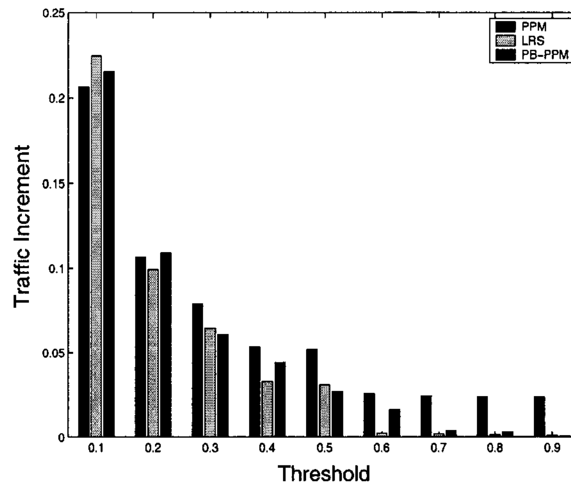


Figure 3.5: Traffic overhead comparisons of three models

Figure 3.5 compares the traffic increment for the three PPM models for predictions of day 46 when the threshold varies from 0.1 to 0.9.

The three models have similar traffic increases when the threshold is less than 0.6. With a 0.2 threshold, for example, traffic increases 10.7, 9.9, and 10.9 percent. With a larger threshold, however, the traffic increase with the LRS PPM model and our model goes down rapidly and closes to 0 with a 0.9 threshold. Overall, the standard PPM model consumes more network bandwidth than the other models. Considering the hit ratios of the three models, our model is the most cost-effective.

Space overhead

Figure 3.6 compares the number of URLs (nodes) that each model stored for predictions about day 46 access. We used the varied number of clients in the day 45 traces to build the prediction model for day 46 prefetching.

The number of nodes that the standard PPM model stores dramatically increases as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

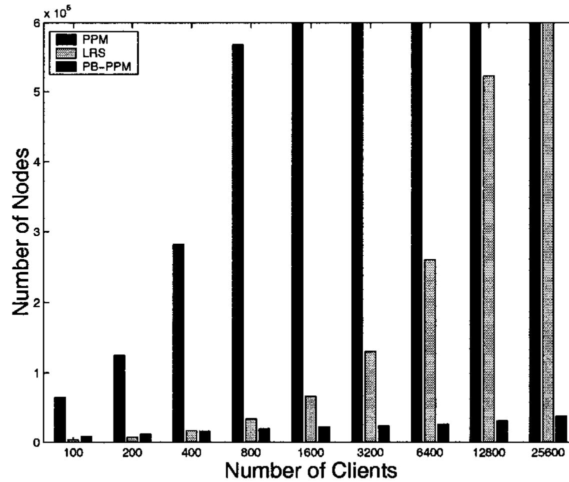


Figure 3.6: Space needed measured by nodes

the number of clients used for prediction increases. The nodes required for the LRS PPM model increases proportionally and quickly with more clients, while the space requirement for our model increases at a much slower pace. With 800 clients for prediction, for example, the LRS PPM stores 71 percent more nodes than our popularity-based model. Using 1,600, 3,200, 6,400, 12,800, and 25,600 clients for predictions, the LRS PPM model stores 1.9, 4.4, 8.7, 15.7, and 26.8 times more nodes than our model.

We see two main reasons for this quick increase. First, the LRS PPM model has many node duplications because it cuts and pastes each tree branch into multiple sub-branches starting from different URLs. Second, as the clients increase, the number of the longest repeating sequences also proportionally increases, but the number of occurrences of subsequences that are also independent sequences decreases. In contrast, the popularity patterns do not change significantly as the client files increase, so our model only moderately increases the number of nodes in the tree structure.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.6 Summary

The popularity-based prefetching technique is an effective Web management approach because Internet storage has become increasingly large and disorganized. Popularity information makes searching and prefetching highly objective and efficient. Our simulation and data analysis revealed two important popularity-related surfing regularities. By building these regularities into the PPM model, Web prefetching can have both high prediction accuracy and a low space requirement.

3.3 Adapting Web Prefetching to Dynamic Server Loads

3.3.1 Overview

With the popularity of World Wide Web, latency perceived by the clients becomes an important factor of the quality of Web services. Web prefetching is proposed to improve the Web access latency. This technique attempts to prefetch to-be-used Web data based on the historical information of surfing activities. Web prefetching is becoming important and demanding, even though Web caching has been widely utilized for the same purpose. The significance of Web prefetching can be summarized as follows. First, accessing proxy caches through a local area network is much faster than accessing Web servers through global area networks. The study in [77] divides the total Web access latencies into internal latencies in local area networks and external latencies in global area networks. Based on case studies, the authors show that the percentage of external latencies in total access latencies is dominantly high. Thus, an effective prefetching for clients based on reference access information can significantly reduce the external latencies. Second, the number and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

types of Web servers have increased and will continue to increase dramatically, providing more services to a wider range of clients. Thus, access variations will keep increasing as more Web servers are emerging [16]. Increasing proxy caching size is no longer the only effective way for performance improvement, because it is mainly beneficial to the commonly shared data objects among clients.

The potential effectiveness of Web prefetching has been widely investigated, and associated overheads have also been noticed. The possible network traffic overhead is analyzed in [48, 118]. It has been shown that if prefetched objects could be transferred at low rates, the network condition would be improved over that without prefetching. In order to avoid network overhead, a partial prefetch scheme [74] and prefetching between proxies and dial-up clients [61] are presented. Recently, researchers propose to utilize the unused network bandwidth for prefetching with marginal effects on existing traffic [76, 115], which makes Web prefetching more practical. The space overhead of building predictor trees could also been reduced by considering the specific access patterns [33, 34, 66, 98]. The use of a threshold to adjust the aggressiveness of prefetching is analyzed in [69]. In contrast to the above cited studies, we look into the performance impact of prefetching and associated overhead in Web servers.

Our research focus on Web servers in this section is motivated by the structure of current Internet services that are heavily rely on HTTP based on TCP protocols. Before an HTTP request is sent to the Web server, a TCP connection must be first established though a three-way handshake mechanism. Once the TCP connection is established successfully, a client can send a series of HTTP requests to a Web server while the server uses the same connection to transfer the requested data to the client. The client-perceived response time

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

comes from three parts: (1) the time to establish the TCP connections; (2) the time for Web servers to processing requests; and (3) the time for response transferred on the network. The last two parts account for the major delay. We further believe the Web server processing time is crucial to ensure the quality of Web services for the following two reasons:

- TCP connection time does not change much when the load on the server changes.

  As pointed out in [90], when the server is lightly loaded, the connection time can be ignored since the processing time is the major part. In fact, Web prefetching is always applied only when the spared resources are available. In our experiments, the average connection time is never larger than 10% of the average client-perceived response time. In order to reduce the connection overhead, `KeepAlive` directive is widely used in HTTP 1.0 and 1.1. In our experiments, we construct the requests with the directive following the format of HTTP 1.0.

- Prefetch requests will not increase the transmission time of regular requests.

  This is because (1) prefetch used for dynamic content does not consume additional network resources; and (2) a new TCP/IP protocol has been proposed [115] to avoid network resource competition between background traffic and existing traffic.

Web prefetching can effectively reduce the server response time since idle server resources can be utilized for Web prefetching activities. Web prefetching technique has been proposed for different kinds of Web services. For static Web objects, a prefetching scheme pro-loads those objects to be accessed possibly in the near future [76]. For dynamically generated Web objects, the server response time can be reduced by pre-generating Web objects based on client access information [109]. For a search engine Web site, performance improvement

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is expected by pre-loading most related searching results [80]. For a CDN provider, pushing related objects to the proper CDN servers can achieve better performance than passively pulling [36]. An ideal prefetching scheme should have no negative effects on existing activities on the Web server while the reduction on client perceived server processing time can be maximized.

However, with the increase in types and quantum of the Web services, the server can easily become a bottleneck in Internet. A major concern about a wide deployment of Web prefetching is related to the associated overhead that may negatively affect the performance of the Web servers and the response time. In this study, we focus on evaluation and and providing solutions to address a major weakness of current Web prefetching — the prefetching activities are scheduled independently of the dynamic server workloads. Therefore, if the prefetching activities are not properly controlled and coordinated with Web servers, the Web access performance can be significantly hurt.

The effectiveness of designing and implementing such a control and coordination mechanism in Web servers mainly rely on insightful understanding and accurately characterizing the dynamic behaviors of Web servers. In this section, we first develop an open queuing model to characterize detailed transactions in Web servers. Using this model, we analyze the server resource utilization and the average response time with different request arrival rates when prefetching is involved different kinds of Web services. Guided by this model, we design a responsive and adaptive prefetching scheme that dynamically adjust the prefetching aggressiveness in Web servers. Our scheme not only prevents the Web servers from being overloaded, it can also minimize the average server response time. We have effectively implemented this scheme on the Apache Web server. Our measurement-based performance

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

evaluation shows the developed model can accurately predict the utilization of Web server resources and correspondent average response time.

3.3.2 Prefetching Performance Analysis

BCMP Queuing Networks

If the customers of a queuing network model have different service demands, it is regarded as a model of multiple class customers. Developed by Baskett *et al.* [18], BCMP queuing networks allow different classes of customers, each with different service requirements and service time distributions other than exponential. Open, closed, and mixed networks are allowed. The queuing networks we developed for prefetching in Web servers are based on an open model, which consists of K devices and C different classes of customers. The network state is denoted by a vector $\vec{n} = (\vec{n}_1, \vec{n}_1, ..., \vec{n}_k)$, where component $\vec{n}_i$ is a vector that represent the number of customers of each class at device i, which is $\vec{n}_i = (n_{i,1}, n_{i,2}, ..., n_{i,C})$. An open network is one in which allow customers to enter or leave the network while a closed network always has a constant number of customers remain in the network.

Queuing Networks for Web Services

In our analysis, we only consider the situation where only a single Web server exists. The results can be easily extended to multiple servers. A typical Web server is connected to a LAN, which is connected to a router that connects the site to the ISP and then to the Internet. The queuing networking model is shown in Figure 3.7. It is a open queuing network model with a queue for each of the three components: network interface card (NIC), CPU and disk.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

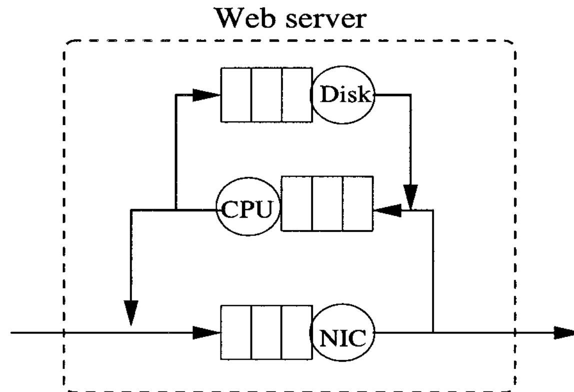


Figure 3.7: The queuing network model for Web services.

A typical Web service consists of several operation steps to be finished. For example, Apache server has the following procedure to process incoming requests:

1. Translating URI to the local filename,

2. Checking ID authorization,

3. Checking access authorization,

4. Access checking other than authorization,

5. Determining MIME type of the requested object,

6. Sending a response back to the client, and

7. Logging the request.

Different steps rely on different devices. For example, the first five steps mainly use the CPU while the sixth step normally needs NIC (network interface card), CPU and disk. With the improvement of Web techniques, a Web server provides various kinds of services to clients. Different kinds of requests are different on resource requirements from each other. When prefetching is applied for a specific class of Web requests, some requests can

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

be prefetched while the rest are still explicitly requested by clients. Prefetching may change the resource requirements. Static object prefetching has a limit on the size of prefetched pages to avoid the overhead of wrong predictions. Dynamic content prefetching utilizes the idle CPU cycles to pre-compute the results which may be requested by clients, but the results are not required to be transferred to clients until they are requested explicitly. Due to the variance of surfing behaviors in the Web, it is natural to model a Web site as an open network with multiple classes requests.

In our analysis, we use BCMP queuing network model to estimate the capacity of the Web server and average server response time. In this section, we give the analysis in a general situation, where the number of devices and number of request classes are not limited. The parameters used in our analysis are shown in Table 3.1.

Parameter	Meaning
K	number of devices in a Web server
C	number of classes of requests
λ_r	request arrival rate of class r
$D_{i,r}$	average service demand of class r requests at device i
$U_{i,r}$	utilization of device i by class r requests
U_i	utilization of device i by all requests
$R_{i,r}$	average response time of class r requests at device i
R_r	class r average response time
R	average response time for all requests

Table 3.1: Input Parameters for Web Service Models

3.3.3 Capacity of A Web Server

Resource Utilization Without Prefetching

We can calculate the utilization of each device by summing the utilization of each class

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of requests:

$$U_i = \sum_{r=1}^{C} U_{i,r} = \sum_{r=1}^{C} \lambda_r D_{i,r}.$$

If a steady state solution exists, we must have

$$max_i \left\{ \sum_{r=1}^{C} \lambda_r D_{i,r} \right\} < 1.$$

It guarantees that no device will receive more service requests than it can handle.

Resource Utilization With Prefetching

When prefetching is applied in the Web server, for a given class of requests, two kinds of requests will be received by the server: *regular requests* are explicitly sent by clients when cache misses happen and *prefetch requests* are automatically delivered by the browser with the prefetching function after it receives the prediction results from the server.

In order to accurately calculate the resource utilization, we divide the class of requests into two parts when prefetching is applied to a specific class of requests.

- λ_r^r: regular request arrival rate of class r after prefetching is applied,
- λ_r^p: prefetch request arrival rate of class r,
- $D_{i,r}^r$: average service demand of regular requests of class r requests at device i, and
- $D_{i,r}^p$: average service demand of prefetch requests of class r requests at device i.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In consequence, additional C new classes of requests will be received by the Web server while the correspondent C original classes of requests may have different resource requirements from those without prefetching. In order to achieve a steady state, the following equation should be followed:

$$max_i \left\{ \sum_{i=1}^{C} (\lambda_i^r D_{i,r}^r + \lambda_i^p D_{i,r}^p) \right\} < 1.$$

3.3.4 Average Response Time

Average Response Time Without Prefetching

In order to compute average server response time of all classes of requests, we need to calculate the average server response time for each class of requests. For class r requests, we have:

$$R_r = \sum_{i=1}^{K} R_{i,r} = \sum_{i=1}^{k} \frac{D_{i,r}}{1 - U_i}.$$

For all classes of requests, the average server processing time is:

$$R = \frac{\sum_{r=1}^{C} R_r \times \lambda_r}{\sum_{r=1}^{C} \lambda_r}.$$

Average Response Time with Prefetching

Since the prediction is based on history information, not all prefetched files are useful. The effectiveness depends on the accuracy of prediction and the prefetch hit ratios. Here are additional parameters we have defined:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- P_r: the prefetch hit ratio of class r customer, *i.e.* the percentage of all requests prefetched before they are requested explicitly by clients,
- A_r: the accuracy of prefetching of class r customer, *i.e.* the ratio between the accessed prefetched files and all prefetched files,
- R_r^r: regular requests of class r response time, and
- R_r^p: prefetch requests of class r response time.

The regular request rate and the prefetch request rate for class r customer can be calculated by:

$$\lambda_r^r = \lambda_r * (1 - P_r),\ \lambda_r^p = \frac{\lambda_r * P_r}{A_r}.$$

The average response time for the two kinds of requests are:

$$R_r^r = \sum_{i=1}^{K} R_{i,r}^r = \sum_{i=1}^{k} \frac{D_{i,r}^r}{1 - U_i},\ R_r^p = \sum_{i=1}^{K} R_{i,r}^p = \sum_{i=1}^{k} \frac{D_{i,r}^p}{1 - U_i}.$$

Considering that the server response time of prefetch requests may not be perceived by clients, we assume all prefetch requests are finished before the clients require them explicitly. We define $client - perceived\ average\ server\ response\ time$ as the ratio between the total server response time of regular requests and the number of requests when no prefetching is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

applied. In order to minimize the client-perceived average server response time, we want to minimize the following equation:

$$R = \frac{\sum_{r=1}^{C} \lambda_r^r \times R_r^r}{\sum_{r=1}^{C} \lambda_r} = \frac{\sum_{r=1}^{C} (1 - P_r) \times \lambda_r \times R_r^r}{\sum_{r=1}^{C} \lambda_r}.$$

A Web server with multiple kinds of service makes the analysis complicated. BCMP queuing model provides a good approximation to estimate the device utilization and response time when multiple classes of requests exist. It also facilitates to account the effects of prefetching on Web servers. By estimating the server resource utilization, we can easily control the prefetching aggressiveness and deduce the average server response time.

3.3.5 Adaptive Prefetching Algorithm

Based on the analysis in previous section, we design an adaptive prefetching algorithm which adjust the aggressiveness of Web prefetching dynamically with the request arrival rates and their service demands.

One important mechanism in the algorithm is to adjust the aggressiveness of Web prefetching to reduce the server processing latency perceived by clients. The procedure of computing average response time of one device is shown in Figure 3.8. The input parameters are all classes of request arrival rates of both regular and prefetch requests and the output is the average response time of the device. Two tables are needed in five different steps.

1. *T, A and P* table shows the relationships among thresholds, accuracies and hit ratios for different level requests of all classes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. *Level Demand* table shows the request distribution in different levels for all classes of requests and correspondent service demands.

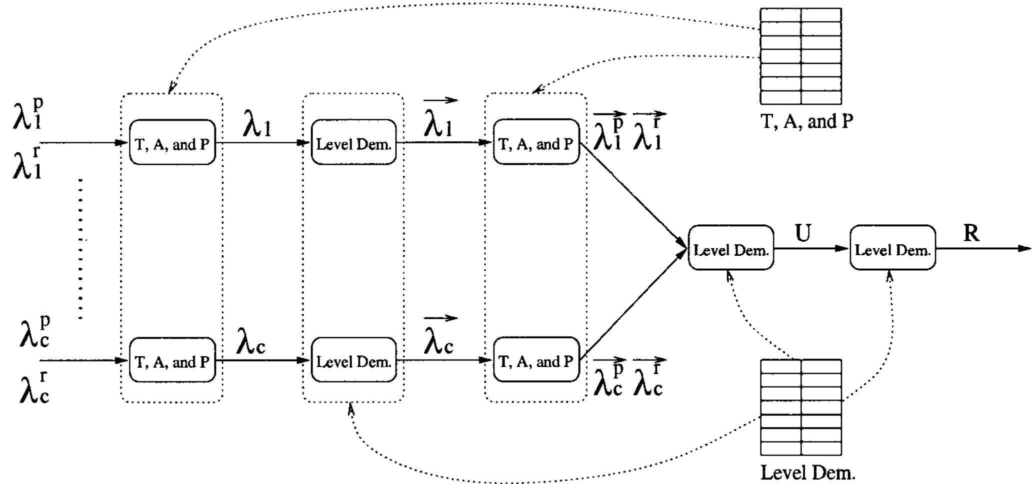


Figure 3.8: The procedure of computing response time of one device

Step 1:

In order to estimate the server response time, we need to know the request arrival rates and the service demands of each class of requests. When prefetching is used, we are not able to observe the request arrival rates directly since part of requests have been prefetched. However, from the previous analysis, for a specific class of requests using prefetching with a given threshold, we can compute the request arrival rates without prefetching by the following equation:

$$\lambda_r = (\lambda_r^r + \lambda_r^p)/(1 - P_r + \frac{P_r}{A_r}).$$

Step 2:

The service demands of each class of requests can be computed by analyzing the server logs or monitoring the server utilization in real time. Due to the observed heavy tail

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

distribution in Web traffic [17, 47], the service demand of the average sized requests is not accurate to represent the whole requests of a specific class. By dividing the requests into several levels by their sizes, we can improve the accuracy when we estimate the whole service demands of the class. If we define $\vec{\lambda_r}$ as the class r requests in different levels, and $\vec{l_r}$ as the percentage of class r requests in different level requests, we have

$$\vec{\lambda_r} = \lambda_r * \vec{l_r}.$$

Step 3:

As we have pointed out, prefetch requests may have different service demands and we need to characterize the request streams including regular and prefetch requests. In our scheme, we also compute the prediction accuracy and hit ratio for each level of requests in a given class. If we know the request arrival rate in each level without prefetching, we can calculate each level request rate of prefetch and regular requests, which are represented by $\vec{\lambda_r^p}$ and $\vec{\lambda_r^r}$.

Step 4:

The total service demands (service utilization) of one class requests using prefetching can be approximated by multiplying the service demand of each level of requests (defined as $\vec{D_r}$) with the request rate of them as follows:

$$U_{i,r} = (\vec{\lambda_r^p} + \vec{\lambda_r^r}) * \vec{D_r}.$$

The device utilization U_i is equal to sum utilization of all kinds of requests on device i.

Step 5:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We can compute the device average response time for each class of requests.

$$R^r_{i,r} = \frac{D^r_{i,r}}{1-U_i},\ R^p_{i,r} = \frac{D^p_{i,r}}{1-U_i}.$$

Furthermore, the average response time of Web server for every class of request can be computed by summing all response time of individual device. The server average response time for all classes of requests can be easily calculated. By repeating the procedure above for all possible thresholds for every class of requests, the optimal value is the one that achieves the minimal server response time for all requests. The description of the whole algorithm can be found in the Appendix.

The workload used in our experiments is from the WorldCup 98 Web site, which is available from the Internet Traffic Archives [8]. It was one of the busiest Web sites in 1998 and represents a popular Web site trace available in the public domain. During the collection period, there were 33 different HTTP servers at four geographic locations, although not all of them were in use for the entire collection period. During this 92 day period (April 26th - July 26th, 1998), 1,352,804,107 requests were received by the Web site. We have conducted our experiments on more than 10 days' traces and all results are consistent. We select the 46th day, one of the busiest days during this period, in our presentation. During that day, a total of 252,753 clients sent 50,395,084 requests for 8,265 data objects on the servers. A total of 187 GBytes was transferred from the servers to all clients. For detailed analysis of the WorldCup workload see [11]. In order to simplify the presentation, in the rest of the section, we only use a single class of requests in our experiments and evaluations.

A heavy-tailed distribution has been observed in Web traffic [17, 47]. A random variable that follows a heavy-tailed distribution varies in a large range of size, many occurrences as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

small mixed with a small amount of occurrences as large. In the Web environment, a large percentage of HTTP requests are for small objects and a small percentage of requests for objects which are several magnitude larger than the small objects.

Request Size Distribution

In current proposed prefetching schemes, the maximum size of the prefetched objects is also set to avoid the high overhead on network if the prefetched objects are not used, which may change the distribution of the size of requested objects. The distributions for different prefetching thresholds are shown in Figure 3.9.

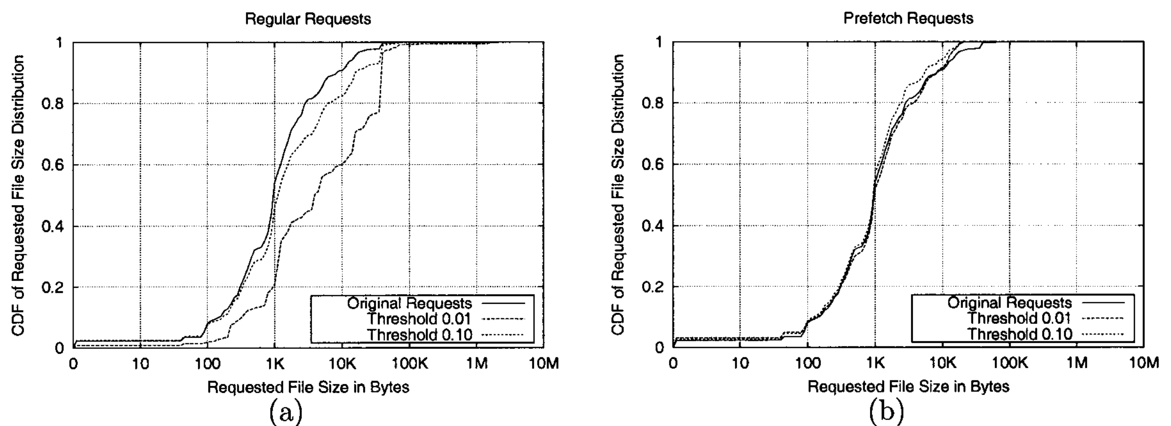


Figure 3.9: (a) The regular request size distributions of no prefetching scheme, and prefetching schemes with thresholds 0.01 and 0.10; (b) The correspondent prefetch request size distributions

The left figure gives the comparison of the size distributions of regular requests among no prefetching scheme and the prefetching schemes with thresholds 0.01 and 0.10. The right figure gives the comparison of the size distributions of prefetch requests in the same experiments. The request size distribution is similar between the prefetch requests and the original requests when prefetching is not used, while the distribution changes a lot for regular requests because the a high percentage of small-sized objects have been prefetched.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In order to accurately compute the device utilization and correspondent average response time, we need to account the changes of the request size distribution.

Classifying All Requests By Size

As pointed out in [47, 84], average results for the whole population of requests would have little statistical meaning due to the large variability of the size of objects. The accuracy of service demands estimation can be improved by dividing the requests into a number of levels by the object sizes.

In our experiments, we also define the maximal size of objects to be prefetched, which should also be considered when categorizing the requests. For the WorldCup 98 traces, the size ranges for different levels are shown in Table 3.2.

Level	File Size Range	Request Percent	Average File Size	CPU	NIC
1	[0 KB, 5 KB)	84.6%	1.1 KB	0.4 ms	0.09 ms
2	[5 KB, 20 KB)	11.9%	10.8 KB	0.8 ms	0.89 ms
3	[20 KB, 100 KB)	3.4%	33.6 KB	1.7 ms	2.78 ms
4	[100 KB, ∞)	0.83%	1149.7 KB	44.2 ms	95.3 ms

Table 3.2: Characterizations of Different Level (Table Level Demand)

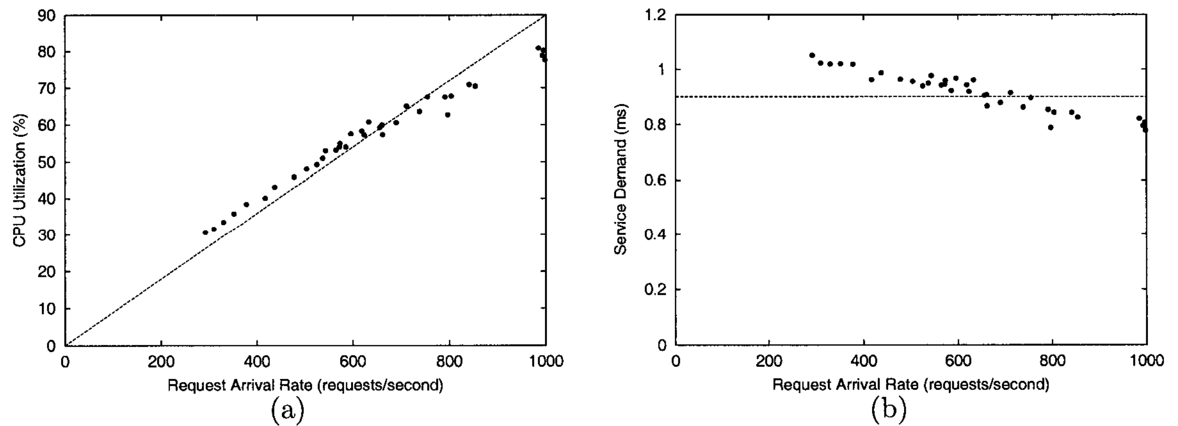


Figure 3.10: The CPU utilization and the correspondent service demands of level 2 requests with different rates

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In order to measure the service demands for every level, we measure the CPU, NIC and disk utilization by changing request arrival rate λ with different parallel connections. The CPU and disk utilization are taken from the Linux `/proc` filesystem and the the NIC utilization is taken using `tcpdump`. In our experiments, we find the disk utilization is marginal and we do not count it in our following analysis. As an example, Figure 3.10 shows the results of the requests in level 2, which is measured by using a PIII 500 MHz computer with 128 MByte memory and a 100 Mbps Ethernet card as the Web server. The left figure shows the CPU utilization with different request arrival rate and the right figure shows the correspondent service demands. The CPU utilization is increased less slowly than the request rate, which results in lower service demands due to more parallel connections. In our experiments, when a single connections is used to send and receive the request, the level-2 requests has higher service demands (about 1 ms). With a large number of parallel connections (larger than 10), the service demands are decreased to 0.8 ms. Considering a busy server connected by a lot of clients, we use 0.8 ms as service demand for the requests in level 2.

	Level 1		Level 2		Level 3		Level 4		Overall	
Threshold	A_1(%)	P_1(%)	A_2(%)	P_2(%)	A_3(%)	P_3(%)	A_4(%)	P_4(%)	A(%)	P(%)
0.01 - 20K	32	93	21	85	/	/	/	/	30	89
0.05 - 20K	44	82	35	71	/	/	/	/	43	78
0.15 - 20K	54	49	49	44	/	/	/	/	54	47
0.25 - 20K	57	24	51	25	/	/	/	/	56	24
0.35 - 20K	62	14	57	13	/	/	/	/	61	13
0.45 - 20K	53	5.9	49	6.7	/	/	/	/	52	5.8
0.55 - 20K	49	2.1	47	3.0	/	/	/	/	49	2.2

Table 3.3: Relationships Among Accuracy, Hit Ratio and Threshold (Table T, A, and P)

As we discussed in the previous section, in order to estimate the prefetch effects, we first

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

need to build a table to collect the accuracies and hit ratios for all possibly used thresholds. Table 3.3 shows the results from traces of day 45, randomly selected from the 92-day period. For those thresholds larger than 0.6, the hit ratios are less than 1% and have very limited influence on the response time. We only focus on thresholds from 0.01 - 0.55.

3.3.6 Prefetching Performance Evaluation

Request Arrival Rate Estimation

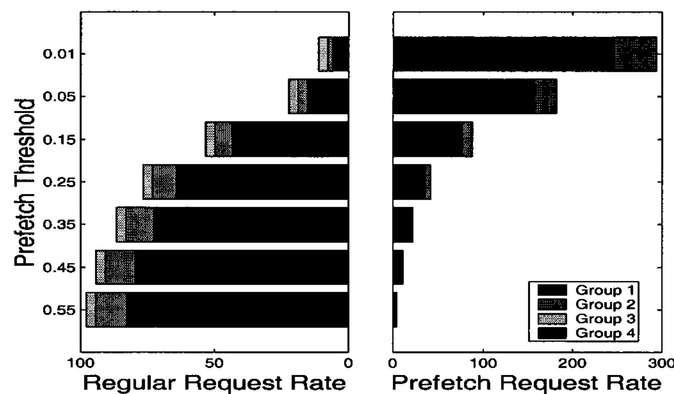


Figure 3.11: The request distributions for each level with different thresholds

In order to evaluate the CPU utilization when prefetching is applied, we need to know the both regular and prefetch request arrival rates, which can be calculated by using Table 3.3. The estimated values for a specific request arrival rate ($\lambda = 100$) are shown in Figure 3.11. It clearly shows that the regular request arrival rates can be effectively reduced by setting low thresholds, while the prefetch request arrival rates is increased very fast. For example, if the threshold is set to 0.01, the regular request rate is reduced to 10 requests/seconds and the prefetch request rate is close to 300 requests/second. Compared with the request arrival rate without prefetching (100 requests/second), the load on the server is increased significantly.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Server Capacity

As we pointed in previous section, the server capacity is determined by the bottleneck device, which is the CPU in our experiments. In order to estimate the CPU utilization, we need to know the request arrival rates in all levels and the service demands of each level request. The estimated server CPU utilization for a specific request arrival rate (100 requests/second) is shown in Table 3.4. As expected, when a low threshold is set, the CPU utilization is increased with the increment of request arrival rates. However, the CPU utilization is increased at a slower pace than the request rate due to a larger percentage of small-sized requests. For example, if threshold is set to 0.01, the request arrival rate is increased from 100 to 305, while the CPU utilization is increased from 5.4% to 15.5%.

Level/Threshold	0.01	0.05	0.15	0.25	0.35	0.45	0.55	No Prefetch
1	251.77	172.86	119.92	99.92	91.86	89.03	86.45	84.6
2	49.96	27.59	17.35	14.76	13.06	12.78	12.3	11.9
3	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4
4	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
Total Request	305.2	203.9	140.8	118.2	108.4	105.3	102.2	100
Demand (ms)	155.1	103.4	73.0	62.7	57.9	56.6	55.1	54.0
CPU Utilization	15.5%	10.3%	7.3%	6.3%	5.8%	5.7%	5.5%	5.4%

Table 3.4: CPU Utilization Comparison among Different Thresholds

Response Time

Once we have the device utilization, we can use the service demands to estimate the average response time of each device. If we assume all prefetched files can be fully downloaded before the clients explicitly request them, the server processing times of prefetch requests are not perceived by clients. Since only part of requests (regular requests) are explicitly sent out by clients, the client-perceived server response time can be reduced after

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

prefetching is deployed. As an example, the number of regular requests and the average client-perceived server response time are shown in Table 3.5 when the request arrival rate is 100 requests/second. For all thresholds in this table, prefetching can always reduce the client-perceived average response time, especially the minimal value is achieved when the most aggressive threshold 0.01 is selected. Although the CPU average response time is increased for each level requests when prefetching is used due to higher CPU utilization, the number of regular requests can be significantly reduced. When the threshold is 0.01, the request rate explicitly sent by the clients is decreased to 11.2 requests/second from 100 requests/second.

	No Prefetch		T=0.01		T=0.05		T=0.15		T=0.25		T=0.35	
	λ	$RT(ms)$	λ^r	$RT(ms)$	λ^r	$RT(ms)$	λ^r	$RT(ms)$	λ^r	$RT(ms)$	λ^r	$RT(ms)$
Level 1	84.6	0.42	5.9	0.47	15.2	0.45	43.15	0.43	64.3	0.43	72.76	3.73
Level 2	11.9	0.85	1.79	0.95	3.45	0.89	6.66	0.86	8.93	0.85	10.35	4.47
Level 3	3.4	1.80	3.4	2.01	3.4	1.90	3.4	1.83	3.4	1.81	3.4	5.97
Level 4	0.083	46.72	0.083	52.31	0.083	49.30	0.083	47.68	0.083	47.16	0.083	66.86
Overall	100	0.56	11.2	0.16	22.1	0.20	53.3	0.35	76.7	0.45	86.6	0.50

Table 3.5: Response Time Comparisons Among Different Thresholds

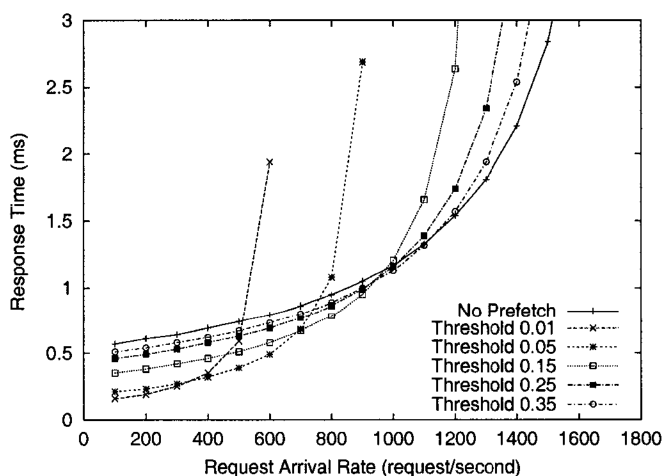


Figure 3.12: CPU response time comparisons among different thresholds

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The response time with variable request arrival rates are shown in Figure 3.12. It is clear that low thresholds should be used when the server's load is light while high thresholds should be set for heavy server's load. It is also interesting to observe that prefetching can not bring any benefits if the request rate is larger than 1000 requests/second. However, most Web servers are utilized far below the maximal capacities to accommodate the bursty request streams. Thus, prefetching can be an effective way in most cases in practice. In our experiments, the response time is normally below 3 ms, which limits the performance improvement of prefetching. If we consider dynamic content with response time of hundreds of milliseconds, prefetching can significantly reduce the response time perceived by clients.

3.3.7 Implementation

We have implemented the proposed prefetching methods on Apache 2.0.40 [2]. The Web server will make predictions for all requests. When it prepares to serve the responses, prediction results will be added in the header and sent back to the clients. When persistent connections are used, a connection can receive both types of requests from the same client. Two kinds of headers have been added in the request: `Regular` and `Prefetch`, which are included in regular requests and prefetch requests, respectively. When more than one previous URL is used to make predictions, the clients also include previous access information with the header. In order to make it compatible with the currently deployed protocols, every request without the additional headers is considered as a regular request. A new header `Prediction` in the server's response header is added to convey the prediction results.

Periodically, the Web server checks if threshold is suitable for current average request arrival rate. A counter is used to record the number of requests received in the last period.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

When the predefined time slice is reached, a maintenance procedure is called. First, it estimates the average request rate in the last period. Then it checks if the current threshold is suitable and selects an optimal one for the current load level. When the request rate is lower than a predefined value, the minimal prefetch threshold value is set safely. For the WorldCup 98 traces, we repeat the procedure every 10 seconds.

Our analysis in previous section assumes that the server knows the client status on current request (hit/miss). It requires an additional message from a client to inform the server when a prefetch hit happens. Our experiments indicate this may significantly increase the overhead on the server. However, if no predictions are made in this situation, the hit ratios can be reduced by 20%. In our implementation, the Web server makes predictions for all incoming requests. When clients receive the prediction results with prefetched objects, they will cache them as well. When those prefetched objects are requested, the client can use the cached prediction results to generate prefetch requests. Although it is not always accurate when more than one previous URL is used, our experiments demonstrate the hit ratio is only 5% lower while up to 40% messages are reduced, compared with the method using an additional message for a hit on browser cache.

Several tools are available to simulate the clients sending requests to Web servers. Instead of testing the capacity of the Web server, we need to replay the request streams from different clients by the original time order. SPECweb99 [10], s-client [14] and WebStone [114] use synthetic workload to benchmark the Web servers, while real clients' traces should be used to test the effects of prefetching on clients' caches, in order to avoid the problems of synthetic workload mentioned in [17]. Another popular tool, httperf [88], which can replay a fixed set of URLs at a given rate, is not suitable for our experiment either, since the time

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

information is important to evaluate a Web server system with prefetching.

The clients are simulated by an enhanced WebStone 2.5. In our enhanced WebStone 2.5, every client process has a URL list recording the requested URLs and the time to send the request, which is extracted from the real Web server traces. In the current implementation of WebStone, the maximum number of `webclient` process is set to 1024, due to the limitation of the number of sockets for a process to open simultaneously. In order to make it scalable, we assign every webclient process several real clients, represented by several URL lists. In this way, we can simulate more than 1024 clients by using a relatively small amount of processes on a limited number of machines.

There are several limits in using multiple processes on one computer. First, the simulation depends on the OS to schedule the processes, which may decrease the burstiness of the original workload. Second, different processes may compete with each other the available network bandwidth. However, these effects are marginal in our experiments for the following reasons:

- Although the number of processes is up to 200 in one computer, the average number of processes to be scheduled is about 20, because the average requests rate every second is about 1/10 of the number of clients. The client-side computers never became the bottleneck measured by CPU and network utilizations.
- In our server traces, the interval of the timestamp is one second, which makes it meaningless to replay the workload strictly by time order.
- Our network interface is pretty fast. Because a very limited number of clients send/receive requests/responses to/from the servers, the competition for network bandwidth has

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

trivial effects on client-perceived latency.

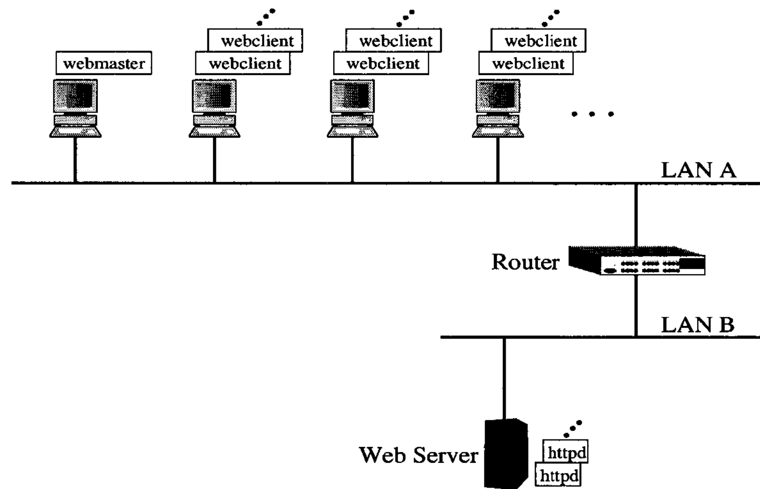


Figure 3.13: The experimental environment

The experimental environment is shown by Figure 3.13, where simulated clients and the Web server are located in different 100 Mbps Ethernet LAN connected by a router. On the client side, a number of clients, which are represented by processes (webclient) distributed on a number of computers, send requests to the server. The `webmaster` is running on another computer to manage the webclient processes and collect the results from all webclients. A number of `httpd` processes are created in the server to process incoming requests.

In our experiments, 100 to 1000 webclients, each in charge of 15 real clients, are equally distributed on 5 computers with Intel 2.26 GHz P4 CPU and 1 GByte memory. The Web server uses a computer with Intel 500 MHz PIII CPU with 128 MBytes memory and a 100 Mbps Ethernet card. The Apache Web server uses the `worker` module to support threads for high performance and uses default parameters in Apache `httpd.conf` to set the initial number of server processes and maximum number of simultaneous client connections. All machines run the Linux operating system with kernel 2.4.18.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

All webclient processes read client traces extracted from traces of day 46 from the WorldCup 98 Web server traces. We use a 10-minute section in the trace of day 46. During the 10 minutes, 15,304 clients visited WorldCup 98 Web site. The cache status has influential effects on hit ratios. In order to make the results more accurate, before we start our experiments, we use a previous hour period trace to warm up the browser caches. The server uses 300,000 requests in the day 46 trace as the training data to build the predictor.

The effectiveness of our adaptive prefetching model is evaluated by two metrics.

- The accuracy of estimating server's capacity. An accurate estimation is important to prevent the Web server from being overloaded.

- The accuracy of estimating server's response time. This value is essential to select the optimal threshold to adjust the aggressiveness of Web prefetching.

In order to make the results clear, we select a normal used threshold 0.05 to present the related results in this section.

Server Throughput

By adjusting the number of clients, a request stream with a variable request rate is used to test the performance of different schemes. Starting from 1,500 clients, an additional 1,500 clients will be added every minute, which results in total 15,000 clients at the end of in the 10 minute test. The request arrival rates and correspondent server throughput for both schemes are presented in Figure 3.14.

In both schemes, the server's throughput is always equal to the request arrival rate until the server's capacity is reached. For the prefetching scheme with threshold 0.05, the server can process up to 2000 requests per second while it can only process up to 1500 requests

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

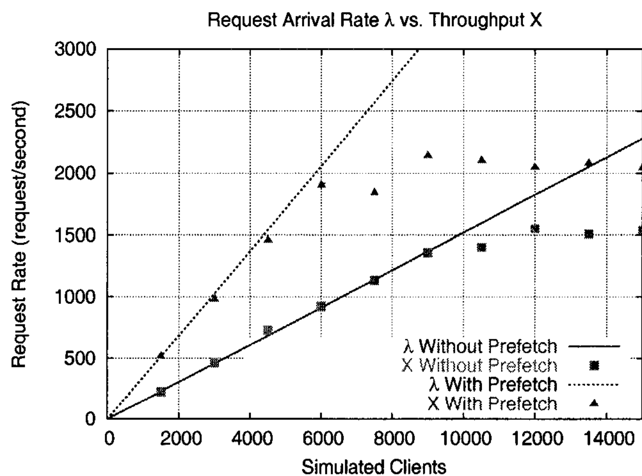


Figure 3.14: The server throughput and request arrival rates of no prefetching scheme and a prefetching scheme with threshold 0.05

per second in no prefetching scheme. There are two reasons: a) the average service demand per request in the prefetching is lower that that in no prefetching scheme. b) the ratio of small sized requests is increased when prefetching is used.

Server Capacity

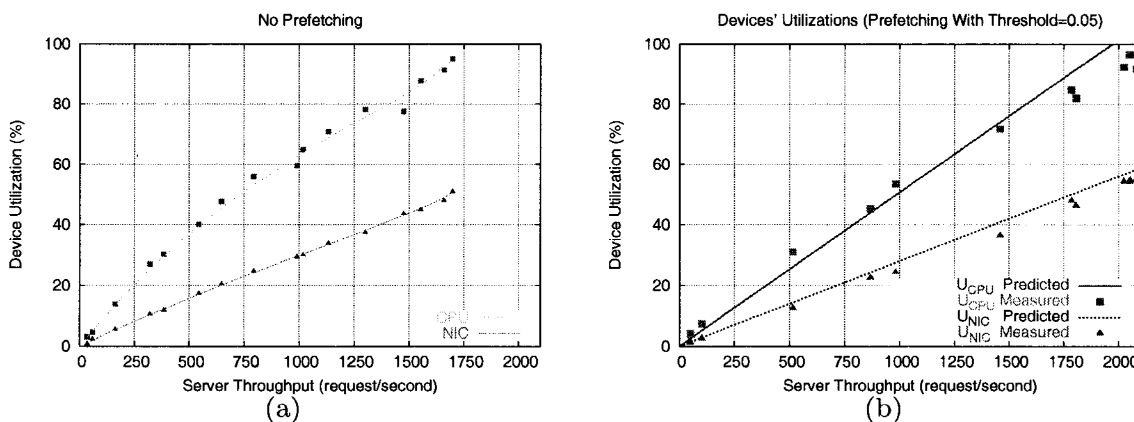


Figure 3.15: (a) The server resource utilizations when no prefetching is used; (b) The comparison of resource utilization between the estimated from the model analysis and measured in our experiments when prefetching is used with threshold 0.05

The server resource utilizations for different server throughputs are shown in Figure 3.15.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The left and right figures present the results for no prefetching scheme and prefetching with threshold 0.05, respectively. With the increase of the server throughput, the CPU utilization is not increased proportionally. When the throughput is approaching the server's capacity, the CPU utilization is increased at a lower pace. As pointed in [47, 84], the service demands can be higher due to the existences of the burstiness in Web request streams. When the request arrival rate is close to the server's capacity, the effects of burstiness on the service demands is reduced. The predicted service utilizations for CPU and NIC are proportionally increased, as shown in the right figure. Our measured results are within 5% from the predicted values.

Average Response Time

The server's response time is the sum of all device response times. In our experiments, the CPU is the bottleneck and the NIC response time is proportional to the server throughput. In Figure 3.16, we present the average CPU response time comparisons between the experimental results and the values predicted by our model. The left figure shows the average CPU response time when prefetching is used. The line is the estimated value by using the queuing model and the points are calculated from our experiment results. When the server's throughput is not very high (less than 1500 requests/second), our model can accurately estimate the response time, while our predicted results become higher than experiment values with the increase of server throughput since the CPU utilization is lower than predicted. The right figure shows the comparison of the average client-perceived CPU response time between the no prefetching scheme and the prefetching scheme with threshold 0.05. The x-axis is the request arrival rate when prefetching is not used. The two lines are estimated average client-perceived CPU response time for no prefetching scheme

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and prefetching scheme with threshold 0.05. The points are calculated value from our experiment results. Prefetching with fixed thresholds 0.05 can reduce the response time for light load (e.g., less than 800 requests/second), while prefetching increases the response time for heavy load. Our predicted results are accurate, which can be used to optimize the prefetching aggressiveness.

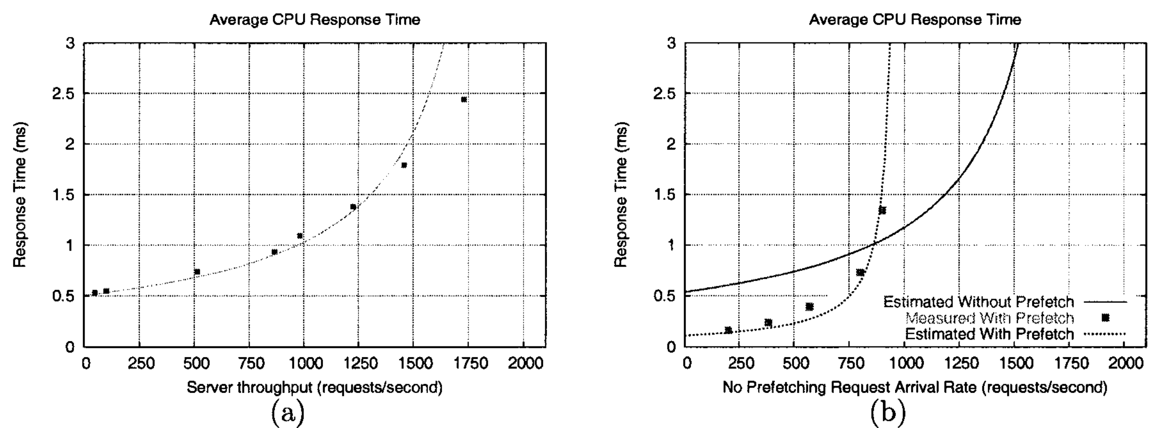


Figure 3.16: (a) The comparison of server response time between the estimated from the model and calculated in our experiments when prefetching is used with threshold 0.05; (b) The comparison of client-perceived server response time in the same experiments

3.3.8 Summary

In this section, we analyze the effects of Web prefetching on Web server's average response time. Although prefetching is well known for its potential to improve Web latency, our study shows it can also increase the Web server response time without proper controls. We have made the following contributions in this study:

- We have developed an open queuing network model to characterize the interactions between prefetching and Web server workloads. The model is validated and proved to be accurate by trace-driven simulations and Web server measurements.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Based on our analysis, we propose an adaptive prefetching scheme to prevent Web servers from being negatively interferenced by prefetching. By monitoring the request arrival rate, the Web servers can adjust the threshold adaptively and periodically to maximize performance.

- We have also effectively implemented our prefetching scheme on an Apache server. The measurement results show that our methods are accurate and responsive, and demonstrates that if the prefetching is used properly, the response time perceived by clients can be significantly improved.

We are currently testing our adaptive prefetching scheme embedded in the Apache server in a real-world Internet environment, where diverse types of Web accesses are conducted, including dynamic and multimedia contents.

3.4 Coordinated Data Prefetching by Utilizing Reference Information at Both Proxy and Web Servers

3.4.1 Overview

Existing prefetching models are either server-based [19, 56, 83] or proxy-based [64, 61]. In a server-based environment, prefetching decisions are made based on the reference access information provided by Web servers. Although Web servers may provide accurate access information, frequent communications between clients/proxies and servers are required, and causing some unnecessary network traffic. In a proxy-based environment, prefetching decisions are made based on the reference access information in proxy servers. Since proxy

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

servers can only predict commonly shared data files among clients, the prediction accuracy can be significantly limited without the input of global Web servers.

In this study, we will address the following three questions: (1) How important is the reference access information in Web servers to data prefetching? (2) Under what conditions is the prefetching accuracy of proxy-based technique sufficient? (3) Can we effectively integrate both server-based and proxy-based techniques to achieve the goal of retaining the prefetching accuracy of the server-based model and minimizing communication between proxy and Web servers?

3.4.2 Evaluation Methodology

Our study and performance evaluation are based on trace-driven simulations. The evaluation environment consists of Web traces, and a simulated Internet environment with clients, proxy servers, and Web servers. We will discuss the selected Web traces and our simulation model in this section.

	requests	objects	clients	servers	successful requests	total Bytes	requests/minute
Day1	22,529,006	6,356,818	66,117	147,577	19,477,134	148,249,338,973	15,645
Day2	22,135,881	6,291,157	66,661	147,054	19,160,640	150,005,649,738	15,372
Day3	23,409,805	6,288,377	65,322	145,988	20,488,491	158,082,426,460	16,256
Day4	22,187,580	6,183,673	66,358	142,338	19,316,066	152,748,081,589	15,408
Day5	21,005,234	6,123,227	59,361	141,869	18,305,089	148,026,687,533	14,586

Table 3.6: Merged Boeing Proxy Traces of 5 Days

	proxy 1	proxy 2	proxy 3	proxy 4	proxy 5	proxy 6
Total Clients	50	200	500	1,000	5,000	10,000
Total Requests	13,566	68,985	154,527	341,619	1,859,045	3,068,912
Successful Requests	12,275	60,613	133,517	302,856	1,607,145	2,653,288
Transferred MBytes	123	360	1,559	2,080	13,353	20,745
Number of Servers	647	1,692	3,482	7,004	24,486	34,179
Number of Files	9,484	321,343	74,958	157,717	679,876	995,879

Table 3.7: Selected Scaled-Down Proxy Traces

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Boeing Company has six proxy servers in its Puget Sound firewall. The first 5 proxies are in a DNS round robin configuration for load balancing. The sixth is running a newer version of software than the others, and is used as a parent for internal proxy cache testing. are directed to this proxy. Since we are interested in tracing accesses of each client, we have used the first 5 traces in our experiments. Table 3.6 presents the characteristics of the 5 merged Boeing traces for 5 days.

One challenge in our study is the lack of correlated server traces and proxy traces accessed by clients. In practice, the available traces are either proxy-based or server-based. Individual server traces are not suitable because we are not able to distinguish accesses between clients and proxy servers. Therefore, we have created pseudo server traces and scaled-down proxy traces from the obtained Boeing proxy traces, which are big enough to build pseudo server traces close to real ones.

We have created multiple proxy trace files with different numbers of clients accessing the proxy. We call them scaled-down proxy traces. The numbers of clients we used are 50, 200, 500, 1,000, 5,000, and 10,000. The clients are randomly selected in the proxy file for each pseudo proxy trace. Since each proxy trace file is a portion of the original Boeing proxy trace file, the trace file contains realistic proxy access reference information. Table 3.7 presents the characteristics of the 6 scaled-down proxy trace files collected from the day trace files on March 4, 1999.

We have created two types of pseudo server traces: *global pseudo server traces*, and *individual pseudo server traces.* For a given scaled-down proxy trace, its global pseudo server trace consists of traces accessing the servers listed in that proxy trace by all the clients who do not use the proxy.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	server 1	server 2	server 3	server 4	server 5	server 6
Total Requests	648,534	243,462	210,131	182,541	157,466	140,473
Successful Requests	647,277	233,216	208,059	181,531	133,854	139,459
Transferred MBytes	1,815	1,204	1,112	875	542	555
Total Clients	21,769	15,193	3,086	2,149	1,810	3,582
Number of Files	5,810	4,185	3,003	3,637	5,130	1,549
Distinct Request Ratio	0.28%	0.40%	0.59%	0.86%	1.75%	0.43%
Distinct File Access Ratio (%)	30.98	22.32	41.00	43.11	45.79	39.12
Concentration ratio (%)	93.79	87.40	93.22	92.43	87.74	90.94

Table 3.8: Selected Pseudo Server Traces

	global 1	global 2	global 3	global 4	global 5	global 6
Total Requests	7,951,627	11,270,833	13,462,240	14,865,900	18,126,906	18,902,854
Successful Requests	6,850,784	9,866,199	11,718,589	12,897,176	15,689,774	16,414,926
Transferred MBytes	38,886	59,204	76,752	82,450	114,371	119,795
Total Clients	63,054	64,769	65,182	65,517	65,987	66,068
Number of Files	1,505,796	1,969,177	2,464,367	2,794,933	3,818,797	4,136,209
Number of Servers	647	1,692	3,482	7,004	24,486	34,179

Table 3.9: Global Pseudo Server Traces

In each of the five merged Boeing proxy trace files, we identified several popular servers that are frequently accessed by clients to form individual pseudo server trace files. Such an individual pseudo server trace is created by collecting all access references to the identified server. We use three invariants to determine whether such a trace is sufficiently realistic to represent a server trace: (1) distinct request ratio (the percentage of distinct requests in the total number of requests), (2) distinct file access ratio (the percentage of distinct files in the total number of distinct server files), and (3) concentration ratio of references (the percentage of requests concentrating on 10% of popular files). In practice, a regular Web server's distinct request ratio, distinct file access ratio, and concentration ratio of references are below 3%, around 30%, and around 90%, respectively [12]. We selected 6 of them for our trace-driven simulations. Tables 3.8 and 3.9 presents the characteristics of the 6 individual pseudo server trace files and 6 global pseudo server trace files collected from the day trace files on March 4, 1999.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The standard 2-order PPM model [39] is used to make predictions in both proxy and servers. The last two access requests will be used to predict the next immediate requests. The threshold is 0.25 through all experiments.

- A Local Area Network (LAN) is constructed between clients and the proxy server. It is a high bandwidth and low latency network with a round trip time, $RTT < 1ms$.
- A Wide Area Network (WAN) is constructed between the proxy server and Web servers. It is a high bandwidth and high latency network with a round trip time $RTT \approx 90ms$.

The procedure of prefetching has two parts: getting prediction results and downloading predicted documents.

Since the clients will send requests for predicted documents after it receive the prediction results from proxy/server, one RTT is needed to approximate the time of getting predicted results.

In order to accurately compute the downloading time of prefetched files, we need to know the available bandwidth and server processing rate. In our simulation, we don't explicitly define the two values. The field *elapsed time* in the proxy trace is used to compute the downloading time. This field records how many milliseconds between a request accepted and send to clients by the proxy, which includes the processing time on the server and transmission time between proxy and server. Based on others' studies, the external latency accounts for more than 80% of all response time. When the prefetched files are cached by the proxy, the downloading time is 1/5 of that between the proxy and the server.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Persistent connection and pipelining requests are also used in our experiments. We assume only one persistent connections between the proxy/client and the server. When a request is issued for predicted documents, it is unnecessary to build another TCP connections, which needs one additional RTT. When there are more than one request for predicted documents, they can be sent to the server without waiting the previous one to be finished due to the existence of pipelining. Using default values of Apache HTTP server version 2.0, we set a fixed holding time of 15 seconds for each persistent connection and a maximum number of requests of 100 per connection. It is possible that *elapsed time* in the proxy log includes the TCP connection time if the server doesn't support persistent connection. We admit it can make our results less accurate.

We assume each client has a browser cache of 10MBytes, which is the default value in currently used browser such as Netscape, and the disk cache size of the proxy is 16 GBytes. Both of them use LRU replacement policy. The value of proxy disk cache is big enough in our simulation because the six proxies we constructed have total traffic from 123 MBytes to 20 GBytes during our test period (1 day). Only proxy 6, having 10,000 clients, needs to make replacement. The value of 16 GB can give us a chance to test the performance when replacement is applied, although it doesn't happen very often. We only prefetch the documents whose sizes are less or equal to 50 KBytes, in order to reduce overhead caused by inaccurate predictions.

3.4.3 Limits of Proxy-Based Prefetching

We first investigated the conditions for a proxy-based prefetching technique to be effective. An ideal server-based scheme assumes that every access from clients to the server can be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

observed by the server regardless of the existence of proxies. A relative hit ratio for a given proxy and Web server is defined as the ratio between a hit ratio obtained by a proxy-based prefetching and a hit ratio obtained by a ideal server-based prefetching. The relative hit ratio presents the relative prefetching ability of a proxy-based prefetching compared with the ideal server-based prefetching.

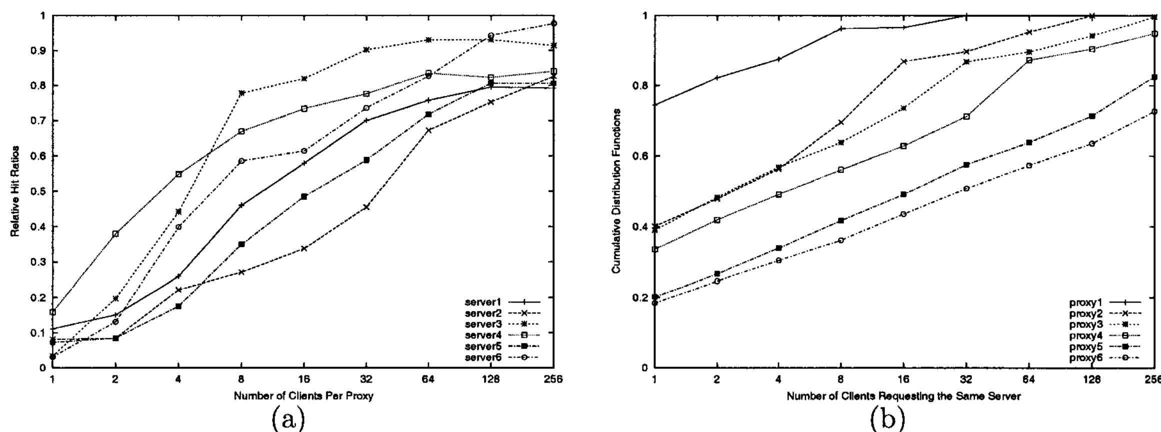


Figure 3.17: (a) The relative hit ratios for accessing each of 6 servers through the proxy where the prefetching is made in the proxy; (b) cumulative density functions (CDF) for the six proxy traces with 50 (proxy 1), 200 (proxy 2), 500 (proxy 3), 1,000 (proxy 4), 5,000 (proxy 5), and 10,000 (proxy 6) clients

Making prefetching predictions at the proxy for the 6 individual servers, we have observed the changes of relative hit ratios by increasing the number of clients to access the proxy (see the left figure in Figure 3.17). We repeated each experiment with an individual server 20 times and presented the average results. This study indicates that proxy-based prefetching ability, measured by the relative hit ratios, increases as the number of clients served by the proxy for accessing a particular server increases. For example, as the number of clients increases to 16 and 64, the average relative hit ratios increase to 59% and 79%, respectively, for accessing each of the six servers from the clients.

The next important question we want to ask is about the shared request distributions

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to different servers through the proxy in a global Internet environment. Studying the distributions, we are able to evaluate the effectiveness and limits of proxy-based prefetching.

The distributions presented in the right figure in Figure 3.17 are measured by the cumulative density functions (CDF) for the six proxy traces with 50, 200, 500, 1,000, 5,000, and 10,000 clients, respectively. The average percentage of the requests consisting of 16 or fewer shared clients going through the proxy to Web servers is 59%.

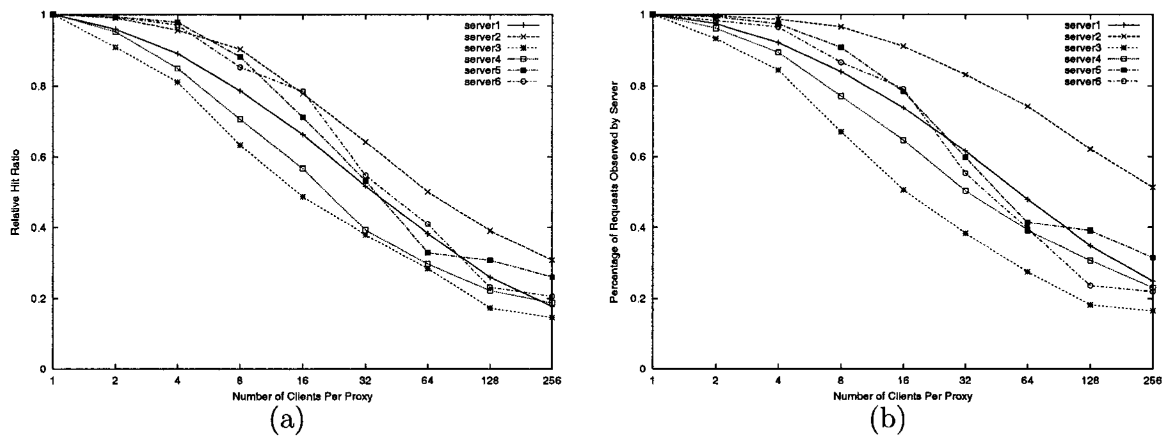
The distribution study indicates that in a normal proxy server with 1,000 clients, proxy-based prefetching can reasonably satisfy less than 40% of the requests by achieving approximately a 60% relative hit ratio. However, for more than 60% of the requests, proxy-based prefetching may not be sufficiently effective. In order to improve the access hit ratios, we should adaptively rely on server-based prefetching for those requests that do not have a sufficient number of shared clients.

3.4.4 Limits of Server-Based Prefetching

With an increasing number of proxies being installed in the Internet to provide caching storage for clients, the burden of a huge number of direct requests to Web servers can be lightened. On the other hand, the accesses directly observed by the Web servers also decrease, which will inevitably weaken the server-based prefetching capability, because this prefetching mainly relies on the quality and quantity of the access information obtained in the Web servers.

We use a relative hit ratio to quantify the prefetching capability, which is defined as the ratio between the hit ratio obtained by a practical server-based prefetching and the hit ratio obtained by the ideal server-based prefetching. In our experiments, we used the same

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) (b)

Figure 3.18: (a) The relative hit ratios for accessing each of 6 servers through the proxy where the prefetching is made in the server; (b) The percentage of requests observed by server through different size proxies

6 individual Web traces. In order to build a predictor tree in a Web server, we limited the number of clients connected to the proxies to 10% of the total clients, and changed the number of clients served per proxy during the experiments.

The left figure in Figure 3.18 presents the changes of relative hit ratios on the 6 servers as the number of clients served per proxy increases. Our trace-driven simulation results show that the server-based prefetching capability is weakened significantly as the number of clients served per proxy increases. For example, as the number of clients per proxy increases from 1 to 16, and to 64, the average relative hit ratios decrease from 100% to 65%, and to 50%, respectively.

We have also measured the changes of the number of requests observed by Web servers as the number of clients served per proxy increases. This evaluation quantitatively shows the trade-offs between lightening the server burden and weakening the server-based prefetching capability. The right figure in Figure 3.18 presents the ratios of number of requests to a server between a practical server-based prefetching and the ideal server-based prefetching

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

on the 6 servers as the number of clients per proxy increases. The ratios also reflect a rapidly decreasing trend of the number of accesses to the server as the number clients per proxy increases.

By comparing the two figures in Figure 3.18, it is interesting to observe that the ability of server-based prefetching measured by the relative hit ration decreases faster than the decreasing pace of the number of requests received by the servers. For example, when the number of clients is 16, the ability of prefetching is about 65% while the number of requests is about 75%. Since the server could not observe all client's requests due to proxy caching, the successive requests from the same client received by the server may not be sequential. However, in order to increase accuracy, prediction algorithms are usually based on several past accesses.(in our experiments, we use two previous accesses to make prediction for next accesses.) In this situation, the accuracy of server-based predictions is a little lower than the ideal scheme.

3.4.5 Coordinated Proxy-Server Prefetching

Prefetching without Coordination

A simple way to overcome the limits of proxy-based and server-based prefetching is to let proxy and server make predictions independently. In this method, the proxy makes predictions for the requests from clients and the server makes predictions for those requests forwarded by the proxy. The proxy and the server do not know the existence of each other. Because the predicted documents are selected by predetermined thresholds, not all predictions would have results. We define a useful prediction as one giving prediction results. An improvement on this simple prefetching is to let the server make predictions

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

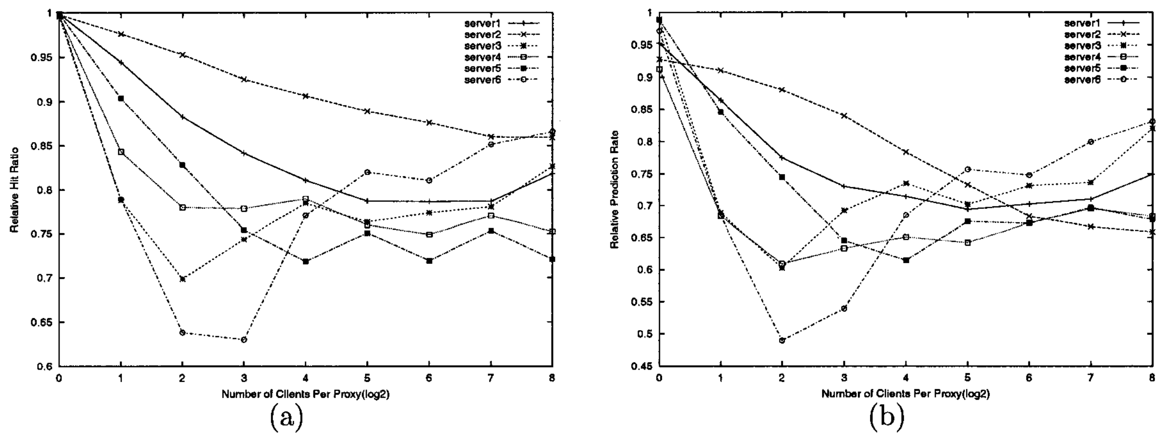


Figure 3.19: (a) The relative hit ratios for accessing each of 6 servers through the simple proxy and server prediction scheme; (b) The relative predictions made by simple proxy and server prediction scheme

when the proxy has no prediction results. We tested the performance of these two schemes by using the 6 individual server traces. We define a relative hit ratio as the ratio between a hit ratio obtained by simple proxy-server prefetching and the improved prefetching scheme.

The results are shown by the left figure in Figure 3.19. The relative hit ratio is decreased as the number of clients per proxy increases first, but it becomes stable when the number of clients is large enough. We observed that when there are more than 4 clients per proxy, the average relative hit ratio is about 80% which means the improved prefetching scheme outperforms simple proxy-server prefetching 25%.

We also measured the number of useful predictions made on both proxies and servers. The right figure in Figure 3.19 shows that the effectiveness of prefetching is proportional to the number of useful predictions, no matter where the prediction is made.

Effectiveness of Coordination

Prefetching will generate overhead on servers. In an ideal server-based prefetching system, a prefetching process will start from a server connection by a client or a proxy after a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

hit or a miss to a requested object in the browser or the proxy. The server will then either send the object along with a list of URLs of predicted objects related to the requested object, if the requested object is missed in the browser or proxy, or just send the list of URLs of objects, if the requested object is hit in the browser or proxy. After receiving the list, the client makes a local search to provide a selected list of URLs of objects that are not available in the browser or the proxy. This list will be sent to the server again. The server will send the selected objects to the client or proxy. The server will be contacted twice by short messages, called packets, from the client or proxy. We compare the numbers of packets used for the server connections between the server-based prefetching and the coordinated proxy-server prefetching. The packet reduction rate is calculated by

$$R_{packets} = \frac{P_{server} - P_{coordin}}{P_{server}},$$

where P_{server} is the number of packets generated from server-based prefetching, and $P_{coordin}$ is the number of packets generated from coordinated prefetching.

The left figure in Figure 3.20 shows packet reduction rates of the coordinated proxy-server prefetching technique over the server-based prefetching system for each of the six individual server traces as the number of clients increases. The reduction rate proportionally and significantly increases as the number of clients increases.

The coordinated proxy-server prefetching technique moves some of the prefetching decisions to the proxy, so that the number of prefetching decisions made in Web servers is reduced. As a result, computing overhead in servers for prefetching can be effectively reduced. The overhead reduction rate is calculated by

$$R_{overhead} = \frac{D_{server} - D_{coordin}}{D_{server}},$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

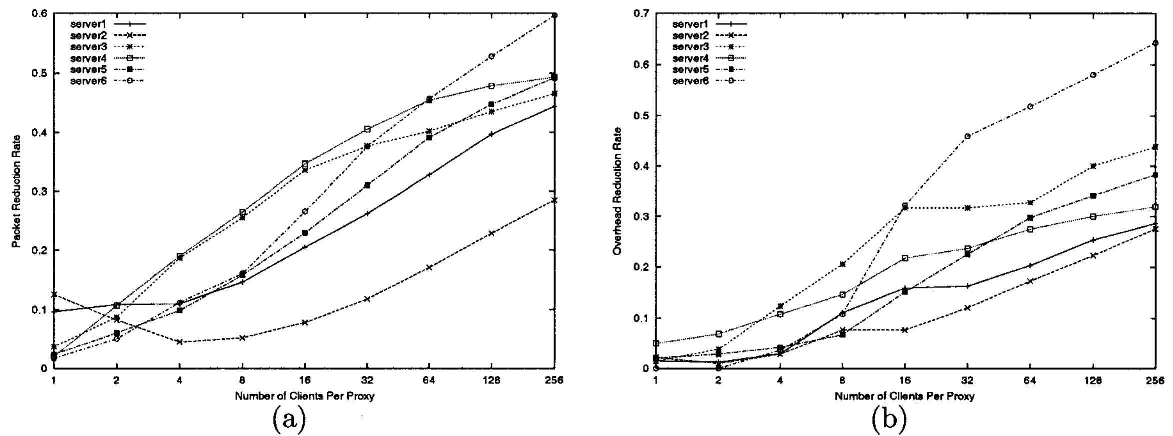


Figure 3.20: (a) The packet reduction rate by using coordinate prefetching compared with server-based prefetching; (b) The overhead reduction rate by using coordinate prefetching compared with server-based prefetching

where D_{server} is the number of decisions made by server-based prefetching, and $P_{coordin}$ is the number of decisions made by coordinated prefetching.

The right figure in Figure 3.20 shows the overhead reduction rates of the coordinated proxy-server prefetching technique over the server-based prefetching system for each of the six individual server traces as the number of clients increases. We show that the overhead reduction rate increases as the number of clients increases.

Coordination Algorithm

Our objective is to adaptively utilize the reference information at both proxy and Web servers. Study([61]) showed the information of access from clients are still very useful for the proxy's predictions. In our algorithm, we still include the input from the clients even when a hit happens in the browser cache. Figure 3.21 presents the outline of the coordinated proxy-server prefetching technique. Upon a client request of an object file, if the request hits in the browser, the object will be accessed locally. The client will also inform the proxy about the access to the object file, and initiate the coordinated proxy-server prefetching

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

process. If the request misses in the browser, the request will be forwarded to the proxy, and the coordinated proxy-server prefetching process is initiated. Starting in the proxy, the coordinated proxy-server prefetching process will handle the following four different cases in the proxy:

1. if the object **exists** and prefetching information related to the object is **available** in the proxy;
2. if the object **does not exist** and prefetching information related to the object is **not available** in the proxy;
3. if the object **exists** and prefetching information related to the object is **not available** in the proxy; and
4. if the object **does not exist** and prefetching information related to the object is **available** in the proxy.

A proxy-based prefetching procedure will handle case 1, while a server-based prefetching procedure will handle case 2. We propose the proxy-server-based prefetching procedures (I) and (II), defined below, to handle case 3 and case 4, respectively. The 4 procedures involved in the coordinated proxy-server prefetching process are shown in Figure 3.21.

server-based prefetching procedure:

(a) The proxy forwards the client request to the server, and asks the server to prefetch related objects based on its PPM model.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

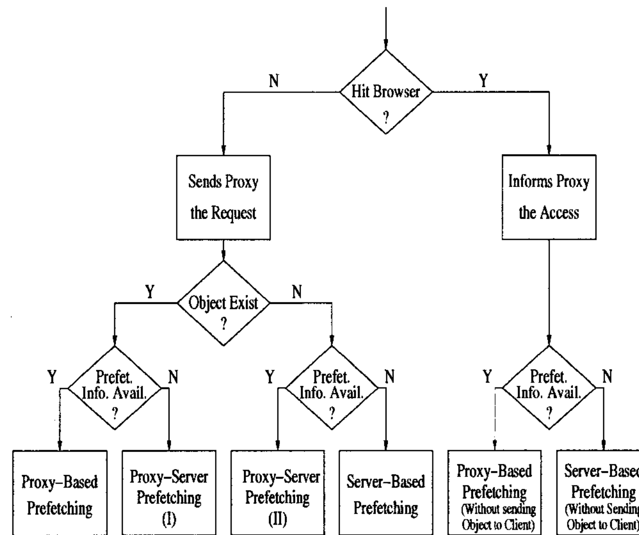


Figure 3.21: The coordinated proxy-server prefetching system design.

(b) The server sends the requested object to the proxy along with a list of URLs of predicted objects.

(c) The proxy stores the object, and sends it to the client along with the list of URLs of predicted objects.

(d) After a local searching, the proxy sends a selected list of URLs of predicted objects to the server.

(e) The server sends the selected objects to the proxy. Meanwhile, the client sends a selected list of URLs of predicted objects to the proxy after a local searching.

(f) The proxy sends the selected objects to the client.

proxy-based prefetching procedure:

(a) The proxy sends the requested object to the client, along with a list of URLs of predicted objects.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(b) After a local searching, the client sends a selected list of URLs of predicted objects to the proxy.

(c) The proxy sends the selected objects to the client.

proxy-server-based prefetching procedure (I):

(a) The proxy sends the object to the client, and asks the server to prefetch related objects based on its PPM model.

(b) The server sends a list of URLs of predicted objects.

(c) The proxy sends the list of URLs of predicted objects to the client.

(d) After a local searching, the proxy sends a selected list of URLs of predicted objects to the server.

(e) The server sends the selected objects to the proxy. Meanwhile, the client sends a selected list of URLs of predicted objects to the proxy after a local searching.

(f) The proxy sends the selected objects to the client.

proxy-server-based prefetching procedure (II):

(a) The proxy forwards the client request to the server, along with a selected list of URLs of predicted objects based on the local prefetching information and a local search.

(b) The proxy sends the selected list of URLs of predicted objects to the client.

(c) The server sends the selected objects to the proxy. Meanwhile, the client sends a selected list of URLs of predicted objects to the proxy after a local searching.

(d) The proxy sends the selected objects to the client.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.4.6 Performance Evaluation

Comparisons of Hit and Byte Hit Ratios

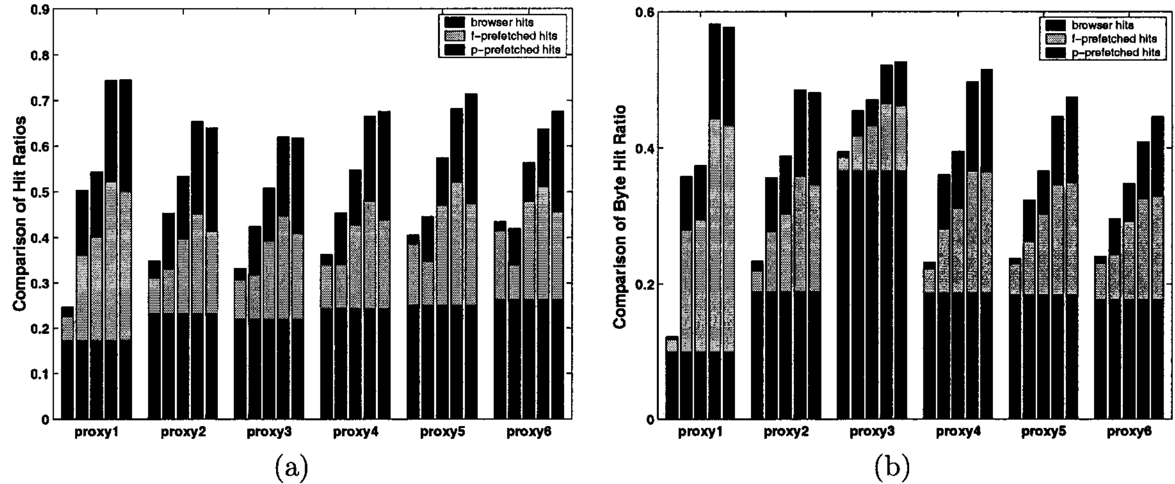


Figure 3.22: (a) The comparisons of hit ratios among the proxy-based, server-based, non-coordinated proxy-server, coordinated proxy-server, and ideal server-based prefetching techniques; (b) The comparisons of byte hit ratios among the five prefetching schemes

Figure 3.22 presents hit ratio comparisons among proxy-based, server-based, non-coordinated proxy-server, coordinated proxy-server, and ideal server-based prefetching techniques using the six proxy traces and their pseudo global server traces. There are three parts of the contribution to hit ratios: the browser cached files, prefetched files and files being prefetching in browser's cache. The left figure in Figure 3.22 shows that the hit ratios contributed from proxy-server-based prefetching are significantly higher than those from proxy-based prefetching, and they are comparable to the hit ratios from server-based prefetching for all the 6 proxy traces. The average hit ratio from coordinated proxy-server prefetching is 75% higher than that of proxy-based prefetching, 50% higher than that of server-based prefetching, 30% higher than that of non-coordinated prefetching, and 3% lower than that of ideal server-based prefetching. By using the proxy to make predictions, the ratio of hits from

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

prefetched files in browser's cache is higher than those without the proxy's participation. It is more effective to reduce latency by applying proxy's prefetch ability properly.

The right figure in Figure 3.22 presents byte hit ratio comparisons. We show that the byte hit ratios are consistent with their hit ratio performance.

Reductions of Global Web Server Loads

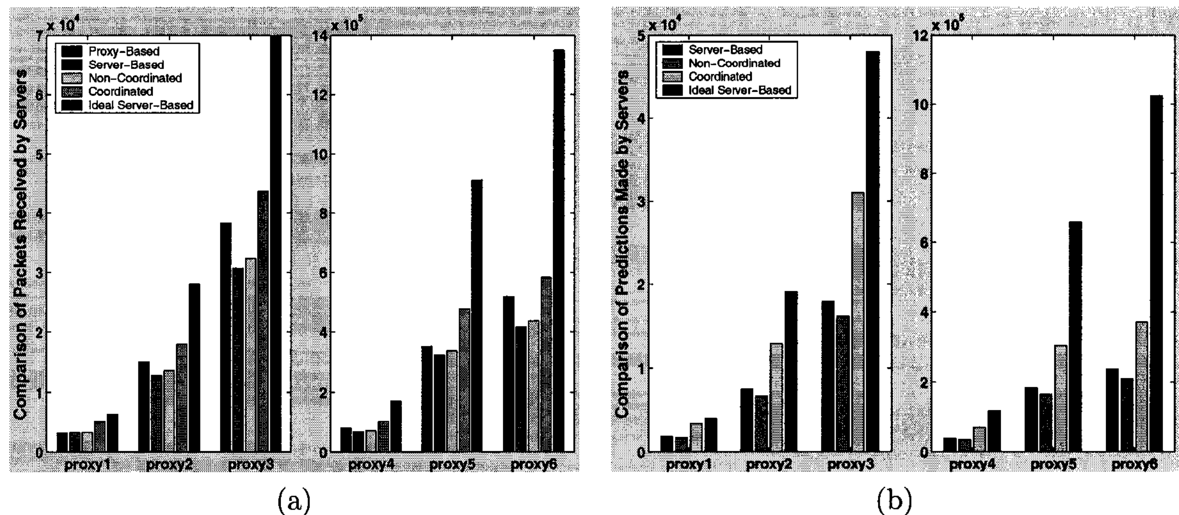


Figure 3.23: (a) The packet reduction rates of the five possible prefetching schemes; (b) The comparison of predictions made on servers of the five prefetching schemes

Using the same trace-driven simulation, we have compared the numbers of packets received by servers, and the numbers of prefetching decisions made by servers for all techniques(no proxy-based scheme in the latter comparison) in Figure 3.23. The trace-driven simulations show that as the number of clients served by the proxy increases, the difference of the numbers of packets sent to servers between the two techniques increases. For example, the packet differences are 434,393 representing a 48% reduction from the server-based prefetching and 766,985 representing a 57% reduction from the server-based prefetching for proxy 5 and proxy 6, respectively. The comparisons of the number of prefetching decisions made by servers follow similar patterns. We should point out that the reduction rates are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

higher than the averages in section 5.2. One of the important reasons is the pseudo traces we built. It is possible that the number of clients observed by servers is not sufficiently larger enough than that observed by proxies, especially when the proxy is large.

Comparisons of Global Traffic and Local Network Traffic

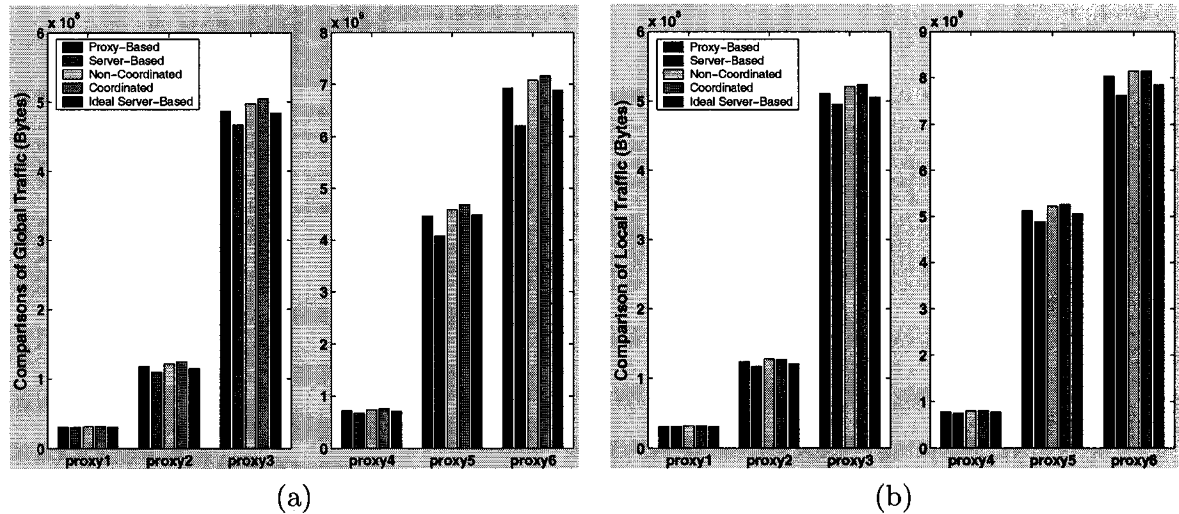


Figure 3.24: (a) The comparison of global traffic of the five possible prefetching schemes for different number of clients; (b) The comparison of local traffic of the five prefetching schemes

Traffic is another important metric to measure the effectiveness of prefetching schemes. Normally, a prefetching method with high accuracy will achieve latency reduction by increasing a small amount of traffic. Due to the existences of proxies, the traffic inside the proxy and outside the proxy are different. We define the traffic outside as global traffic, which is measured as the total transferred bytes between proxies and servers. The traffic inside is defined as local traffic, which is measured as the total transferred bytes between proxies and clients.

The left figure and the right figure in Figure 3.24 shows the comparison of global traffic and local traffic for 5 prefetching schemes respectively. There is no large difference among the schemes (within 10%). Coordinated prefetching and Non-Coordinated prefetching needs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a little more traffic than other methods. It is noticeable that the proxy-based scheme's traffic is almost equal to ideal server-based prefetching, which means the accuracy of the predictions made by proxies is lower than those made by servers.

3.4.7 Summary

In this study, we have investigated the issues of coordinations between proxy-based prefetching and server-based prefetching. We show that the reference access information of servers is important to data prefetching, but should be utilized effectively. Our trace-driven simulations show that we have effectively integrated and coordinated both server-based and proxy-based techniques to achieve the goal of retaining the prefetching accuracy of the server-based model and minimizing the communications between proxy and Web servers.

3.5 Final Remarks

This chapter presented several techniques to improve the deployability of Web prefetching.

1. We proposed a popularity-based prefetching model. By ranking the URLs by relative popularity, we can divide all URLs into different grades. It improves the conventional PPM methods by considering surfing regularities into building predictor trees. Our popularity-based model can achieve much higher hit ratio while the structure is much more space-effective. This design is for general-purpose, and can widely benefit Internet prefetching deployment and applications.

2. We designed an adaptive prefetching scheme in which the prefetching threshold is dynamically adjusted by the request arrival rate. In this method, we divide the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

request rate into different ranges and estimate the response time for each level when different thresholds are used. When request rate is in one of the predefined range, the appropriate threshold will be selected. This adaptive prefetching scheme addresses the concerns of prefetching overhead. With this scheme, prefetching is no longer an independent activity, but a coordinated effort with other activities of servers, such as Web services, avoiding the negative effects of prefetching.

3. We analyzed the limits of proxy-based prefetching and server-based prefetching. Furthermore, we proposed coordinated prefetching by utilize the information from proxy caches and Web servers. Our coordinated prefetching can significantly reduce the server overhead while maintaining high performance. The technique addresses the issue on effectively and globally utilizing the access information for prefetching.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

DNS Consistency

4.1 Introduction

The Domain Name System (DNS) is a distributed database that provides a directory service to translate domain names to IP addresses for millions of Internet sites. [86], [87]. DNS consists of a hierarchy of nameservers, with thirteen root servers at the top. For such a hierarchical system, caching is critical to its performance and scalability. To determine the IP address of a domain name, the DNS resolver residing at a client sends a recursive query to its local DNS nameserver. If no valid cached mapping exists, the local DNS nameserver will resolve the query by iteratively communicating with a root server, a Top-Level Domain (TLD) server, and a series of authoritative DNS nameservers. All the replied DNS messages including referrals and answers are cached at the local DNS nameserver, so that subsequent queries for the same domain name will be answered directly from the cache. Therefore, DNS caching significantly reduces the workload of root and TLD servers, lookup latencies and DNS traffic over the Internet.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

However, DNS only supports weak cache consistency by using the Time-To-Live (TTL) mechanism. The TTL field of each DNS resource record indicates how long it may be cached. The majority of TTLs of DNS records ranges from one hour to one day [72]. While most of the domain-name-to-IP-address (DN2IP) mappings are infrequently changed, the current approach to coping with an expected mapping change is cumbersome. Among numerous DNS related RFCs, only RFC 1034 [86] briefly describes how to handle an expected mapping change: "If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency during the change, and then increased back to its former value following the change."; but the RFC does not specify how much and in what magnitude the TTL value should be reduced. The propagation of the mapping change may take much longer than expected. This pathology is induced by some local DNS nameservers that do not follow the TTL expiration rule and violate it by a large amount of time [92]. More importantly, there are unpredictable mapping changes due to an emergency, such as terror attacks or natural disasters, in which the loss or failure of network resources (servers, links and routers) is inevitable [60]. We have to direct the related Internet services to alternative sites, since people do need service availability at that crucial moment. Furthermore, the widely-deployed dynamic DNS solution, which provides prompt IP mapping for each Web site using a temporal IP assigned by Dynamic Host Configuration Protocol (DHCP), makes the association between a domain name and its corresponding IP address much less stable.

Therefore, without strong cache consistency among DNS nameservers, it is cumbersome and inefficient to invalidate the out-of-date cache entries. In addition, Internet services are likely to be lost because of cache inconsistency. During the cache inconsistency period, the clients served with out-of-date mappings cannot reach the right Internet hosts. An

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

aggressively small TTL (on the order of seconds) can lower the chance of cache inconsistency, but at the expense of significant increase of the DNS traffic, name resolution latency and the workload of domain nameservers [110]. Moreover, the TTL-based DNS redirection service provided by Content Distributed Networks (CDNs) only supports a coarse-grained load-balance, and is unable to support quick reaction to network failures or flash crowds without sacrificing the scalability and performance of DNS [92].

In this chapter, we propose a proactive DNS cache update protocol, called *DNScup* to maintain strong cache consistence among DNS nameservers and improve the responsiveness of DNS-based service redirection. The core of *DNScup* uses a dynamic lease technique to keep track of the local DNS nameservers whose clients are tightly coupled with an Internet server[1]. Upon a DN2IP mapping change of the corresponding Internet server, its authoritative DNS nameserver proactively notifies those local DNS nameservers still holding valid leases. While maintaining strong cache consistency among DNS nameservers, dynamic lease also minimizes storage overhead and communication overhead. Based on service importance to their clients and client query rates, it is the local DNS nameservers themselves that decide on whether or not to apply for leases (or renewal) for an Internet service. On the other side, the authoritative DNS nameserver grants and maintains the leases for the DNS resource records of the Internet service. The duration of a lease is dependent on the DN2IP mapping change frequency of the specific DNS resource record.

While strong cache consistency may be optional for general Internet services, *DNScup* is essential to provide the critical or popular Internet services having always-on service

[1]Either the clients frequently visit the Internet server or the services provided by the Internet server is critical to the clients.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

availability. In addition to maintaining cache coherence among DNS nameservers, *DNScup* can also be used to improve the responsiveness of DNS-based network control as suggested in [92]. Also, we can apply the functionality of *DNScup* to maintain state consistency between a DNS nameserver of a parent zone[2] and the DNS nameservers of its child zones, in order to resolve the lame delegation problem [94].

Based on the DNS dynamic update protocol [103], we build a *DNScup* prototype with minimized modifications to current DNS implementations [58, 87]. Our trace-driven simulation and prototype implementation demonstrate that *DNScup* achieves strong cache consistency of DNS and significantly improves its performance and scalability. Note that *DNScup* is backward compatible with the TTL mechanism, and can be incrementally deployed over the Internet. Those local DNS nameservers without valid leases still rely on the TTL mechanism to maintain weak cache inconsistency.

The remainder of this chapter is organized as follows. Section 4.2 presents our DNS dynamics measurements. Section 4.3 gives a detailed description of the proposed *DNScup* mechanism. Section 4.4 evaluates the *DNScup* based on the trace-driven simulations. Section 4.5 presents the prototype implementation of *DNScup*. Finally, we conclude the chapter in Section 4.6.

4.2 DNS Dynamics Measurement

The purpose of our DNS dynamics measurement is to answer the question of how often a DN2IP mapping changes. In general, a mapping change may cause two different effects. If

[2]Zone is a delegated authority unit that is a manageable domain name space.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the original DN2IP mapping is one-to-one, then the change may lead to the loss of Internet services. We classify this kind of change as a physical change. However, if the original DN2IP mapping is one to many, the changes may be anticipated to balance the workload of a web site as CDN does. We classify these changes as logical changes.

To examine the DN2IP mapping change behaviors, one possible way is to use `dig` to contact remote nameservers directly. However, we observe that only about half of authoritative DNS nameservers allow direct communication with remote resolvers. Therefore, we set up a local DNS nameserver using Bind 9.2.3 [6] to generate probing DNS queries for a collection of Web sites (more than 15,000). We purge our local cache every time we probe a Web site, in order to guarantee that each response comes from an authoritative DNS nameserver instead of the local cache. All measurement experiments were conducted between November 30, 2003 to January 3, 2004. In the rest of this section, we describe the DNS resource record classification and the collection of domain names. Also we present a technique to differentiate the domains using CDNs, in which most of mapping changes are logical changes, from the domains where most of mapping changes are physical changes, including those using dynamic DNS. According to the affiliated Top-Level Domain (TLD) and the popularities, we further categorize the domains into several groups. Then, we measure the TTLs of their DNS resource records and investigate the effect of domain popularity upon DNS TTL behaviors. Based on the measured TTLs, we choose the appropriate sampling resolution to detect the DN2IP mapping changes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2.1 DNS Resource Record Classification

The various mappings in the DNS name space are called resource records. The most widely used resource records include `SOA` records (authority indication for a zone), `NS` records (authoritative name server reference lists for a zone), `A` records (domain name to IP address mappings), `PTR` records (IP address to domain name mappings), `MX` records (mail exchangers for a domain name), and `CNAME` records (alias to canonical name mappings). A type `A` record provides the standard domain name to IP address mapping, while the other type records like `NS`, `CNAME` and `MX` records are used as references. In most cases, more than one authoritative DNS nameserver and mail exchanger serve the same zone due to reliability concerns. Among these DNS resource records, the type `A` record is the most popular record being queried, accounting for about 60% DNS lookups on the Internet [72].

Any type of resource records listed above may change for various reasons. For example, the primary master DNS nameserver within a zone may increase the serial number in `SOA` records to keep the records of the zone's slaves updated; `NS` and `MX` records need to be updated if any authoritative DNS nameserver or mail exchanger is renamed; `A` and `PTR` records need to be changed if the domain name is either renamed or mapped to a different IP address; changes on `CNAME` records have already been utilized by CDN providers to redirect a client request to different surrogates. Note that CDN providers and popular Web sites rotate different `A` records with small TTLs for the same domain name to balance the workload of Web servers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

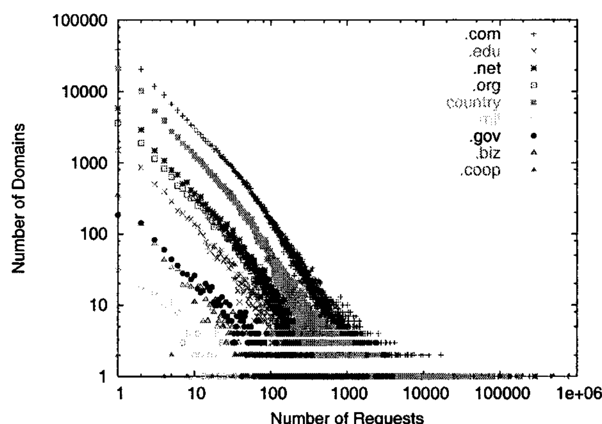


Figure 4.1: The regular domain name distribution with the number of requests in each groups

4.2.2 Domain Name Collection and Grouping

Since DNS is predominantly used by Web sessions to resolve the IP addresses of Web sites, our measurements are focused on the dynamics of the mappings between Web domain names and their corresponding IP addresses. We collected the Web domain names from the recent IRCache [7] proxy traces between November 6, 2003 to November 12, 2003. All Web domain names are classified into three categories: domains using CDN techniques, domains using dynamic DNS techniques, and the rest of collected domains. We refer them as CDN domains, Dyn domains, and regular domains, respectively. Because most CDN domains and Dyn domains include specific text strings to indicate the names of their providers (e.g., `Akamai` for CDN domains, `DynDns.com` for Dyn domains), we can distinguish those domains from the regular ones by the specific strings. In our measurement experiments, we examined 23 major CDN providers [3] and 95 major dynamic DNS providers [4].

Due to the large number of regular domains we collected, the regular domains are further divided into nine groups with respect to their Top-Level Domains (TLDs). They are ended with `.com`, `.edu`, `.net`, `.org`, `.mil`, `.gov`, `.biz`, `.coop`, and country codes, respec-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tively. The regular domain name distribution with the number of requests in each group is plotted in Figure 4.1. As shown in Figure 4.1, most regular domain names fall into the following five major groups: `.com`, `.net`, `.org`, `.edu`, and country domains. Each group consists of three sub-groups:

- popular domains (with the number of requests being larger than or equal to 100 in our one week trace[3]);
- normal domains (with the number of requests being less than 100 but larger than or equal to 10 in one week trace); and
- unpopular domains (with the number of requests being less than 10 in one trace).

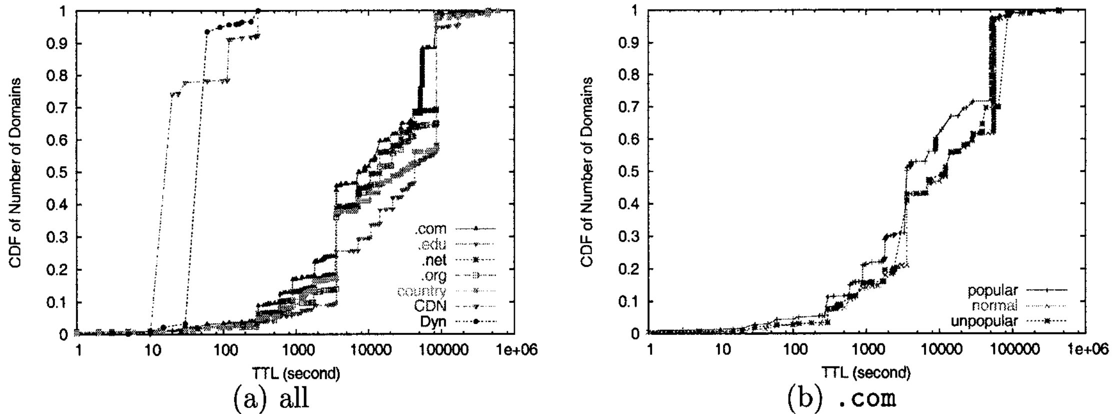
We select 1000 domain names from each sub-group of the five major groups, except for the popular one of `.edu` group where we only have 514 domains available. Note that not all domain names in our regular domain groups follow the strict one-to-one mapping between domain names and IP addresses. Some domain names may use `CNAME` to avoid the direct use of CDN/dynamic DNS providers.

4.2.3 TTLs' Distribution

Different domain names have different TTL values for caching their DNS replies. The TTL distribution of all measured domains is shown in Figure 4.2 (a). For CDN domains, the majority of TTLs have the values of 20 or 120 seconds. For Dyn domains, the majority of TTLs have the values of 30, 60, or 90 seconds. For regular domains, the majority of TTLs

[3]The limited client space and the hidden load factor of caching reduces the number of requests we have seen.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) all (b) .com

Figure 4.2: TTL distributions: (a) All kinds of domain names; (b) `.com` domain names.

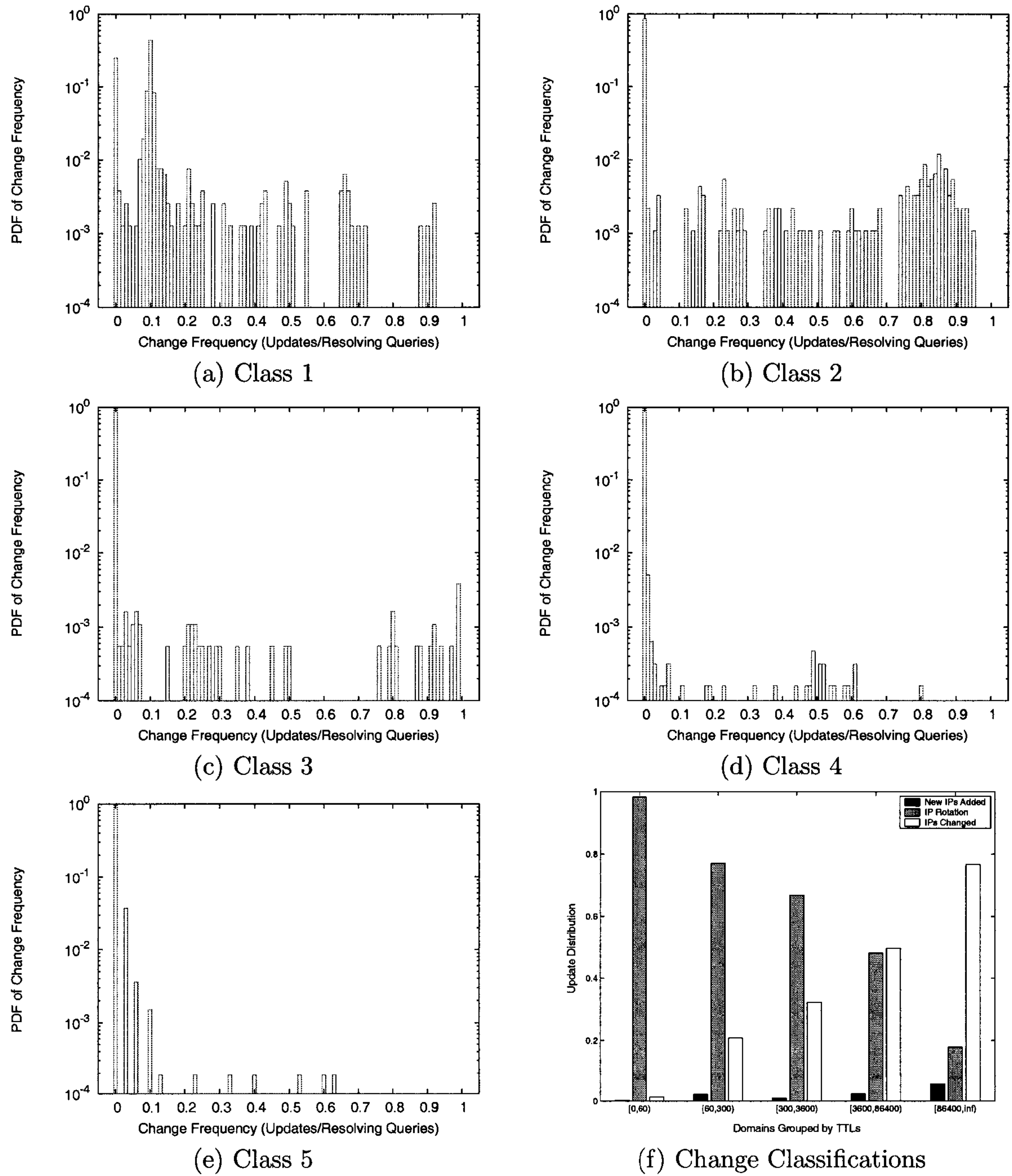
have the values of 300, 3600, or 86400 seconds. The TTL distribution of `.com` domain names with different popularities is shown in Figure 4.2 (b). The TTL distributions for other kinds of domains with different popularities are similar to that of `.com`. We observed that the TTL of a domain name is independent of the domain's popularity.

Table 4.1: Measurement Parameters

Class	TTL (s)	Resolution (s)	Duration
1	[0,60)	20	1 day
2	[60,300)	60	3 days
3	[300,3600)	300	7 days
4	[3600,86400)	3600	7 days
5	[86400,∞)	86400	1 month

The sampling resolution of detecting a DN2IP mapping change is highly dependent upon the values of TTLs. On one hand, our sampling resolution for a specific Web domain should be at least as small as its TTL, in order to capture every possible change that could cause cache inconsistency. On the other hand, to minimize the impact of probing DNS traffic, our sampling resolution should be set as large as possible. Based on the measured TTLs' distribution, we set different sampling resolutions to detect DN2IP mapping changes at different Web sites. The sampling resolutions with respect to the range of TTLs are listed in Table 4.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Class 1

(b) Class 2

(c) Class 3

(d) Class 4

(e) Class 5

(f) Change Classifications

Figure 4.3: The DN2IP mapping change for each class with different TTLs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2.4 Measurement of Mapping Changes

Each domain name in our collection is periodically resolved to check if the mapping has been changed. Depending on the sampling resolution, the duration of a measurement experiment varies from 1 day to 1 month. According to the sampling resolution, the Web domain names being probed in our measurements can be divided into five classes as shown in Table 4.1. Since all CDN and Dyn domains' TTL values are bounded by 300 seconds, they belong to either class 1 or 2. The regular domains of each TLD may fall into all five possible classes, because of the wide spectrum of their TTLs.

A DN2IP mapping change is detected when the responses of two consecutive DNS probes for the same domain name are different from each other. We define the relative change frequency of a domain name as the ratio between the number of mapping changes we detected and the total number of DNS probes we sent for that domain name. The absolute change rate is the product of relative change frequency and the reciprocal of sampling resolution. For ease of presentation, we employ relative change frequency as the metric to study the dynamics of DN2IP mapping changes, and simply call it change frequency in the rest of this chapter. Note that the sampling resolution varies among different classes. Given the same relative change frequency, the corresponding absolute change rates under different classes are different.

The change frequencies for five different classes are shown in Figures 4.3 (a), (b), (c), (d) and (e), respectively. Based on the DNS probing results, we identify three causes that lead to the DN2IP mapping changes: (1) a domain name is relocated to a different IP address; (2) the available IP addresses for a domain name are increased; and (3) the IP address of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a domain name rotates around a set of IP addresses. The first cause results in physical changes, while the second and third causes result in logical changes. The distributions of the changes due to different causes are shown in Figure 4.3 (f) for all five classes.

Physical Changes. As shown in Figures 4.3 (c), (d) and (e), the domains in classes 3, 4, and 5 rarely change their DN2IP mappings, with about 95% domains in these classes remaining intact. Moreover, those domains that have changed their DN2IP mappings have very low change frequencies. For instance, in class 5, almost all changed domains have their change frequencies below 10%, which means a change happens every 10 days. On average, the change frequencies are about 3%, 0.1%, and 0.2% for the domains in classes 3, 4, and 5, respectively. This implies that the average life times of DN2IP mappings are 2.5 hours, 42 days, and 500 days, respectively. However, as shown in Figure 4.3 (f), nearly 40% mapping changes in class 3 and the majorities of mapping changes in classes 4 and 5 are physical changes. Any physical change could cause a cache inconsistency, leading to a loss of service availability. Considering the large number of domain names in classes 3, 4 and 5, the probability of a physical change happening per minute is close to one. Therefore, maintaining strong cache consistency is essential to avoid loss of connection availability.

Logical Changes. The DN2IP mappings in classes 1 and 2 change frequently. In class 1, more than 70% domains changed their IP addresses during a one-day measurement. Most changed domains have their change frequencies around 0.1. In class 2, only about 20% domains changed their IP addresses during a three-day measurement, but most changed domains have relatively high frequencies (e.g., 0.8). On average, the change frequencies of classes 1 and 2 are about 10% and 8%, much higher than the previous classes. The average life-times of DN2IP mappings are 200 seconds and 750 seconds in classes 1 and 2,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

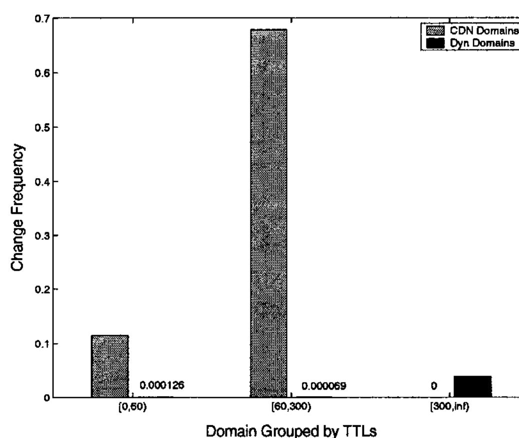


Figure 4.4: CDN and Dyn domain change frequencies with different TTLs.

respectively. As shown in Figure 4.3 (f), such frequent changes are mainly due to IP address rotation (e.g., CDN's load balancing over multiple hosts), and most of the DN2IP mapping changes are logical ones. The more detailed change frequencies of CDN and Dyn domains are illustrated in Figure 4.4.

As shown in Figure 4.4, CDN domains have very high change frequencies: 10% with TTLs between 0 and 60 seconds; and close to 70% with TTLs between 60 and 300. Two major CDN providers dominate the domains of the two ranges: Akamai with TTL 20 seconds; and Speedera with TTL 120 seconds. The domain names served by Akamai have change frequencies around 10%, while those served by Speedera have change frequencies close to 100%. In contrast to CDN domains, the Dyn domains have a low mapping change frequencies: 0.4% with TTL larger than or equal to 300 seconds; and close to zero with TTL less than 300 seconds. Compared with the change frequencies of CDN and Dyn domains, the corresponding TTLs are set aggressively low, resulting in up to 10 and 25 times more DNS traffic than necessary. This redundant DNS traffic would be significantly reduced if server-initiated notification service of *DNScup* were used.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

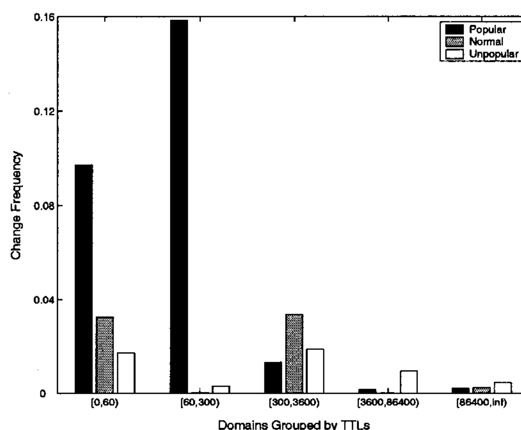


Figure 4.5: The change frequencies of .com domains with different popularity and TTLs.

Within each TLD domain group, we investigate the relationship between DN2IP mapping change frequencies and domain popularities. The measurement results of `.com` domains are shown in Figure 4.5. The results of other TLD domains are similar to those of `.com`. In classes 1 and 2 (most changes are logical changes), we observe that a more poplar a domain tends to have a higher change frequency than a less poplar domain. This is because a popular Web site tends to use CDN or dynamic DNS techniques in order to improve its scalability and performance. By contrast, in classes 3, 4 and 5 (most changes are physical changes), there is no strong correlation between change frequencies and domain popularities. A partial reason for this is that the the occurrence of mapping changes in these classes is sporadic — irregular and random — over the entire domain space.

The resolving latencies for each class are also measured in our experiments. On average, about 8% of domains in each class take more than 1 second to be resolved, which is consistent with the results from [72]. Note that the resolving latency of class 1 is shorter than the rest of classes. This is counter-intuitive, because CDNs use two-level DNS nameservers [4]

[4]A query from a client is first redirected to the CDN provider's high-level nameservers. The high-level

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for location-aware content distribution , which normally needs more resolving time. In our experiments, many domains in this class share the same high-level name server, whose NS record is cached after resolving the first DNS query. Consequently, the subsequent resolving procedures do not need to access the high-level nameservers. This is also consistent with one of major conclusions of Jung's study [72].

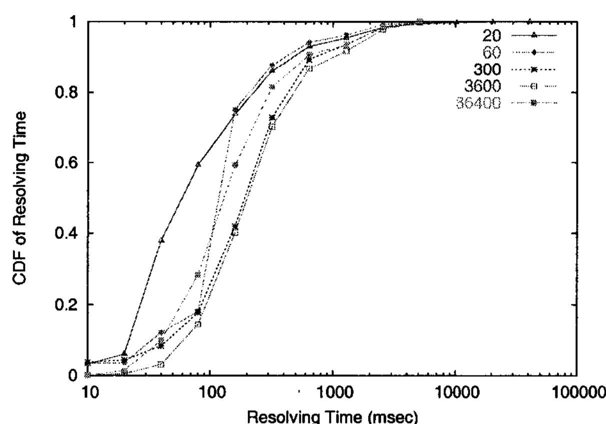


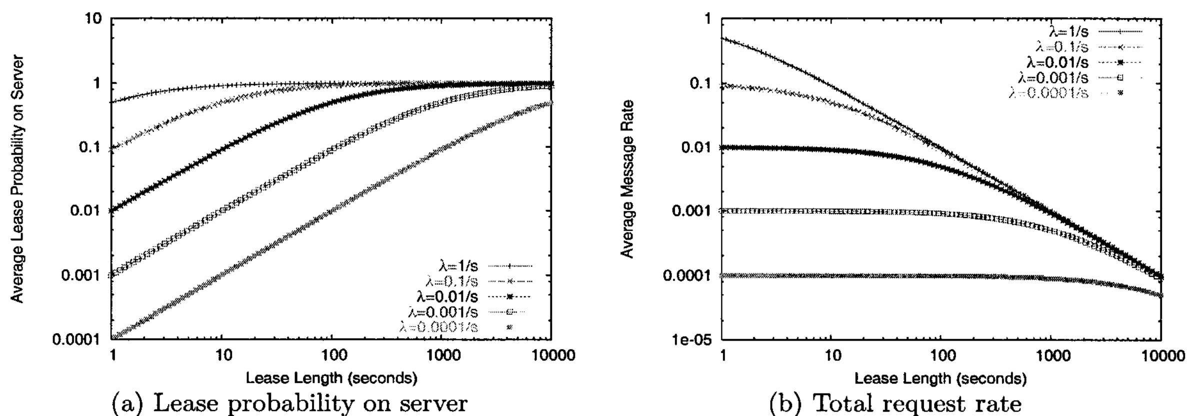
Figure 4.6: The resolving latencies for each class with different TTLs.

4.3 DNS Cache Update Protocol (*DNScup*)

Basically, *DNScup* consists of three components, including mapping change detection module, state-tracking module and update notification module. The mapping change detection module is straightforward to implement, since only the authoritative DNS nameserver has the privilege to change a DNS resource record. There are two ways for an authoritative DNS nameserver to change a DNS resource record: one is through manual reconfiguration and the other is through the DNS dynamic update command such as `nsupdate`.

nameserver is responsible for selecting a low-level nameserver based on the client location, then the low-level nameserver decides the specific server to achieve load balancing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Lease probability on server (b) Total request rate

Figure 4.7: The space and message changes for fixed length lease schemes.

The update notification module is in charge of propagating update notifications. To reduce communication overhead and latency, we choose UDP as the primary transport carrier for update propagation. TCP is used only when a firewall is set on the path from the authoritative DNS nameserver to a DNS cache. Also, we employ timers, retransmissions, and acknowledgment mechanisms to achieve reliable communication for cache updates. When a nameserver has sent a cache update notification message but has not yet received the corresponding acknowledgment, it retransmits the message three times before aborting. The timer is doubled at each expiration.

The core of *DNScup* is the state-tracking module, which keeps track of the recent visitors, i.e., the other DNS nameservers who query and cache a local resource record recently. In the rest of the section, we detail our design on this module, and then we present the whole working procedure of DNScup.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.1 Design Choices

In general, there are three different approaches to maintaining strong cache consistency: adaptive TTL, polling-every-time and invalidation. The major challenge of using TTL approach to maintain cache consistency lies in the difficulty of setting an appropriate time-to-live value for a record. Adaptive TTL [30] handles the obstacle by adjusting the TTL based on the predictions of record lifetime, which has been applied in Web caching consistency management [31]. Adaptive TTL may keep the staleness rate very low, but it cannot support strong cache consistency. The polling-every-time approach is a simple strong consistency mechanism, which validates the freshness of the cached content at the arrival of every request. However, its fatal drawback lies in the poor scalability as shown in [82], incurring many more control messages, higher server workload and longer response time. The invalidation approach relies on the server to notify the clients when an update happens, which is efficient when objects are rarely updated. Because most DNS resource records are changed at very low rates, server-driven invalidation is an appropriate approach to maintaining strong cache consistency among DNS nameservers.

Lease [63] is a variant of server-driven invalidation mechanism. A lease is a contract between a server and a client [5]. During leased period, the client is promised to receive an invalidation notification if a leased object is changed. However, if the client does not have a lease or the lease has already expired, the server must validate a cached object upon the arrival of a request. The lease mechanism is thus a combination of polling and invalidation approaches. A critical question in applying a lease mechanism is how to choose

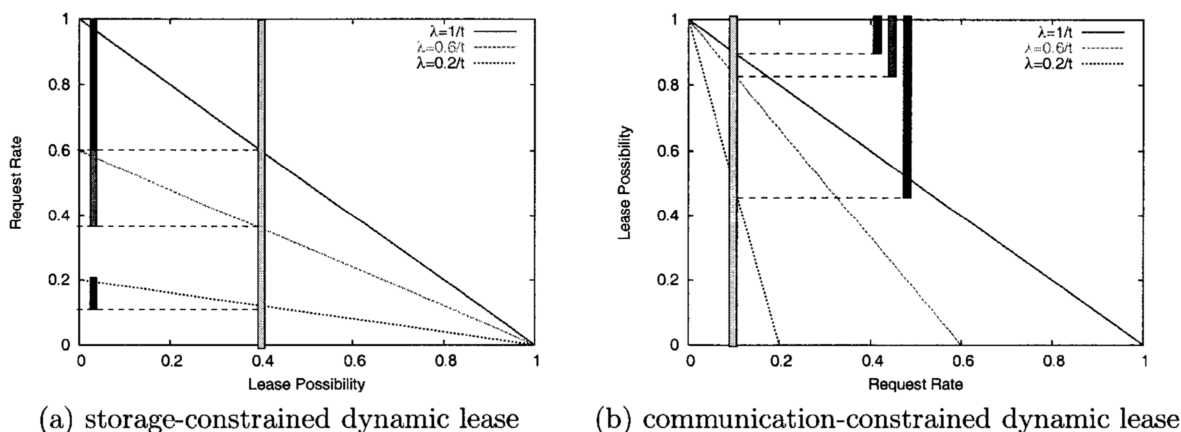
[5] In the context of DNS, the client of an authoritative DNS nameserver is just a local DNS nameserver or another authoritative DNS nameserver that queries the authoritative DNS nameserver

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the appropriate length of a lease. A long lease increases server storage and the number of invalidation messages, while a short lease increases the number of object requests and lease renewal messages.

A lease contract becomes valid either (1) upon the arrival of a new client request if the current lease expires, or (2) by the automatic renewal of an expired-to-be lease. The resultant performance difference lies in the server storage overhead and the client-perceived latency. Because the most DNS resource records do not change often, minimizing consistency maintenance cost is more important than reducing latency. In our study, we always use the first approach to reducing server storage overhead.

To maintain strong cache consistency, *DNScup* requires the authoritative DNS name-server to keep track of the recent visitors that access and cache a DNS resource record. The *recent* in this context implies that the cached record should have not expired yet in these visitors' caches. To make the presentation easier to understand, we refer to these recent visitors as DNS caches in the rest of the chapter. We design a **dynamic lease** scheme to balance DNS server storage requirements and DNS traffic between the authoritative DNS name server and the DNS caches.



(a) storage-constrained dynamic lease (b) communication-constrained dynamic lease

Figure 4.8: Example: dynamic lease with different constrains.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.2 Lease Length Effectiveness

Lease storage overhead on the authoritative DNS nameserver is represented by the probability of the server holding a lease for each DNS cache. Its upper bound is 1, indicating that the server always keeps a lease for a DNS cache. The message overhead is represented by the query rate between the server and its DNS caches. If the lease length is much shorter than the lifetime of a resource record, most messages will be renewal requests from DNS caches and only very few invalidation and update messages may be observed. In the following analyses, since our practical algorithms always set the maximal lease length be much smaller than the resource record lifetime, the communication overhead incurred by invalidation and update messages from the server can be ignored.

We assume that the request arrivals for a DNS resource record follow a Poisson distribution with average arrival rate of λ. The rationale behind this assumption is two-fold: (1) a DNS resolution precedes the beginning of a session communication; and (2) Floyd and Paxson [97] have shown that the session-level (like FTP and Telnet) arrival rate still follows a Poisson distribution, although the packet arrival rate is non-Poisson.

Since the time interval is exponentially distributed, the length of duration between two contiguous leases is equal to the average interval of two contiguous requests, $\frac{1}{\lambda}$. Suppose the authoritative DNS nameserver grants a fixed-length lease, t, at the arrival of a request. The expected probability for the server to maintain the lease, P, is thus

$$P = t/(t + \frac{1}{\lambda}). \tag{4.1}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The lease renewal message rate is

$$M = \frac{1}{t + \frac{1}{\lambda}}. \tag{4.2}$$

Figure 4.7 (a) shows the dynamics of the probability P under different query arrival rates, while the lease length varies from 1 to 10000 seconds. Figure 4.7 (b) shows the dynamics of the correspondent average message rate.

Theorem 1 Assume the request arrivals for a DNS resource record follow a Poisson distribution with average request rate λ. The ratio between the message rate reduction and the increase of lease probability is a constant, which is equal to λ.

Proof: Suppose the lease length is increased from t_1 to t_2. For a given request rate λ, the increase of lease probability on the server is:

$$\triangle P = t_2/(t_2 + \frac{1}{\lambda}) - t_1/(t_1 + \frac{1}{\lambda}) = \frac{\lambda t_2 - \lambda t_1}{(\lambda t_1 + 1)(\lambda t_2 + 1)}$$

The reduction of message rate is:

$$\triangle M = 1/(t_1 + \frac{1}{\lambda}) - 1/(t_2 + \frac{1}{\lambda}) = \lambda * \frac{\lambda t_2 - \lambda t_1}{(\lambda t_1 + 1)(\lambda t_2 + 1)}$$

The ratio between the increase of the lease probability and message rate reduction is equal to λ.

■

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.3 Dynamic Lease Algorithms

Assuming the overhead allowance (storage or communication) is pre-defined, we propose two dynamic lease algorithms: one minimizes the communication overhead given a constraint on storage budget; and the other minimizes the storage overhead, given a constraint on communication traffic. Whether or not a lease is signed between the server and a DNS cache is based on the DNS cache's request rate, while the length of a lease is determined by the DN2IP mapping change rate at the server.

Storage-constrained Dynamic Lease

We define the storage overhead allowance as the maximal number of valid leases that a server can manage. Given the storage overhead allowance P_{max}, the storage-constrained dynamic lease algorithm minimizes the message exchanges for signing and keeping the leases at the server.

Suppose that a total of N DNS resource records $R_i(i = 1...N)$, are maintained on the authoritative DNS nameserver, each with maximal lease length $L_i(i = 1...N)$. Each record R_i is requested by n_i DNS caches $C_{ij}(j = 1...n_i)$, with the request rate λ_{ij}. Our objective is to determine the appropriate lease length of a resource record for each DNS cache l_{ij}, in order to minimize the overall communication overhead M_{all}. The decision should be made under the following constraints:

- for the record R_i and DNS cache C_{ij}, the lease length l_{ij} should be within the range of 0 and L_i.
- the total storage consumption P_{all} should be less than the predefined storage overhead allowance P_{max}.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thus, the consistency maintenance problem can be defined as below:

$$\text{minimize } M_{all} = \sum_{i=1}^{N} \sum_{j=1}^{n_i} M_{ij}$$

$$\text{subject to for any } R_i \text{ and } C_{ij}, 0 \leq l_{ij} \leq L_i$$

$$P_{all} = \sum_{i=1}^{N} \sum_{j=1}^{n_i} P_{ij} \leq P_{max}$$

A consistency maintenance scheme that fulfills the above constraint is a feasible solution. We refer this kind of optimization as the storage-based lease problem (SLP). Since SLP is equivalent to a Knapsack problem, it is NP-complete, but its optimal solution can be found by utilizing the greedy algorithm.

If we have multiple records with different maximal lease lengths, we need to sort the $\frac{\triangle M_{ij}}{\triangle P_{ij}}$, each of which is equal to λ_{ij} based on *Theorem 1*, and then we grant the leases to the DNS cache with the highest request rate. If the server always grants leases with their maximal lengths to the DNS caches selected as above until reaching the storage constraint, we can guarantee that the total request rate covered by leases is maximal.

As an example, suppose that we have three DNS caches with different request rates at 1, 0.6, and 0.2, respectively. The dynamics of request rates with respect to the lease probability is shown in Figure 4.8(a). Clearly, when we increase the lease probability close to the storage constraint, we should grant the lease to the DNS cache with the largest request rate, in order to reduce communication overhead. Note that we ignore the cache update messages in calculating the communication overhead, however, we have proved the greedy algorithm above is still optimal if the cache update overhead is included [32].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Communication-constrained Dynamic Lease

Similarly, given the communication overhead allowance, we can design an algorithm that minimizes the storage overhead. It is also a *NP-complete* problem, and we employ the greedy algorithm to find the optimal solution. Different from the storage-constrained dynamic lease, at the beginning of the algorithm, all DNS caches of a record are granted with the maximum-length leases. After that, we select the DNS cache with the smallest query rate and deprive its lease. This selection and deprivation continue till the communication allowance is satisfied. In this way, we can guarantee the remaining DNS caches have the minimal storage requirements on the server.

Let us consider a scenario in which we have the same number of DNS caches as in the previous example. The dynamics of the lease probability with respect to request rates is shown in Figure 4.8(b). If the request rate allowance is 0.1, we choose the DNS cache that has the lowest request rate of 0.2 and deprive its lease. Thus, we can achieve the maximal storage reduction.

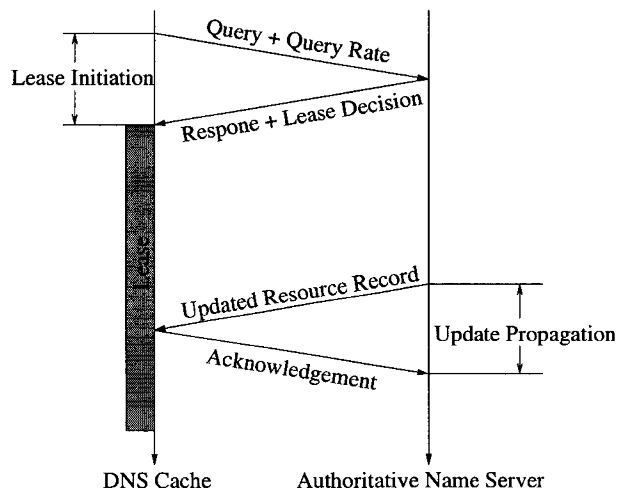


Figure 4.9: DNScup Procedure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.4 Working Procedure of DNScup

Figure 4.9 illustrates the working procedure of DNScup. There are two major communication processes in this procedure: lease initiation and update propagation. The lease initiation is prompted by a DNS cache sending a query to the authoritative DNS nameserver. The query includes the local request rate on the cache as well as its domain name. The authoritative DNS name server evaluates the request rate by a certain metrics (e.g., storage or communication constrain) to make a decision on granting a lease to the DNS cache or not. If a lease is granted to the DNS cache, the authoritative DNS nameserver records the IP address of the DNS cache and the queried resource record. The decision on granting lease is piggybacked to the DNS cache with the response of the query.

The authoritative DNS name server initiates the update propagation when one of its resource records has been changed. Notification messages, containing the updated resource record, are sent to the DNS caches with valid leases. All notified DNS caches need to acknowledge the receipt of the update message. The following two auxiliary functions are important to DNScup.

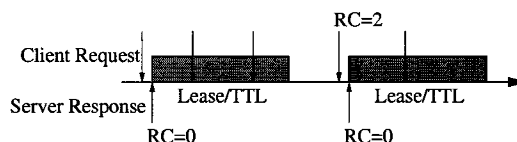


Figure 4.10: DNScup Cache Reference Counter

- *Monitoring Request Rate at the DNS Cache*: In order to measure the request rate for a cached resource record, the DNS cache uses a reference counter (RC) to record the number of requests during a resource record's lease (or TTL period if no lease is signed yet). After the cached resource record expires, the DNS cache book-keeps the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

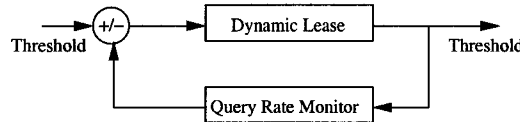
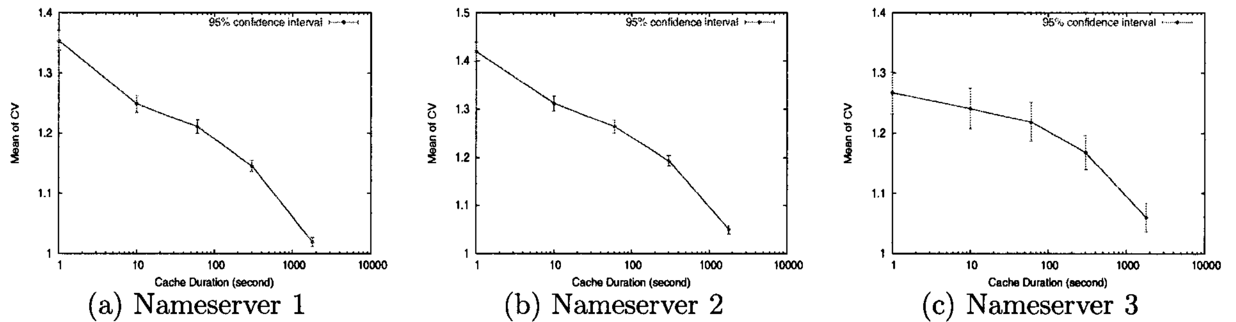


Figure 4.11: DNScup Server Lease Threshold



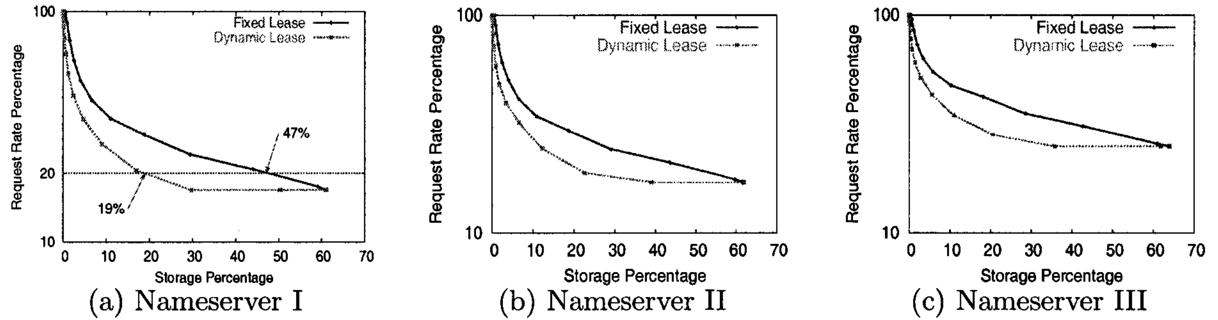
(a) Nameserver 1 (b) Nameserver 2 (c) Nameserver 3

Figure 4.12: The mean of CV of query interval in DNS traces

RC with the domain name by either writing into a specific file or keeping it at the cache for a certain period. When the resource record is queried again, the number of requests during previous lease (or TTL) will be retrieved and forwarded to the authoritative DNS nameserver. Upon the arrival of the new response from the server, the counter will be reset. Figure 4.10 illustrates the usage of reference counter.

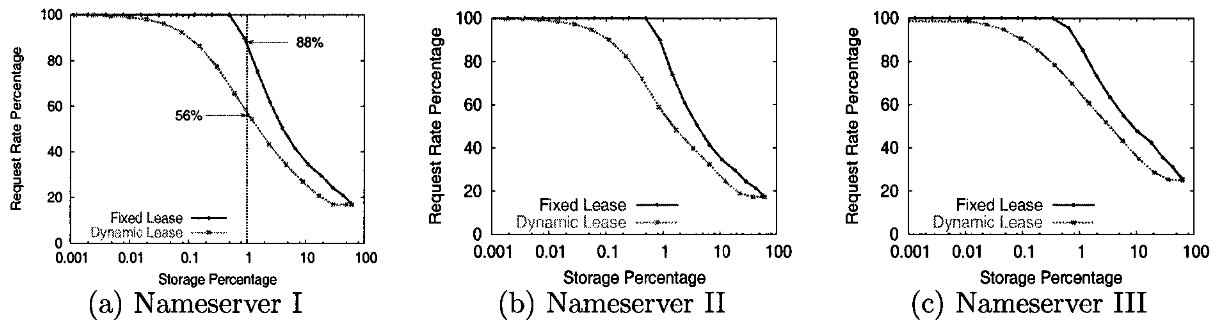
- *Granting Leases in the Authoritative DNS Nameserver*: Using a dynamic lease algorithm, DNScup sets a threshold on cache query rate to determine whether or not the DNS nameserver should grant a lease for a DNS cache. The dynamic lease algorithm can be either evoked periodically to recompute the threshold or kept running to adjust it on-the-fly. In both designs, a query rate monitor maintains the statistics of all related cache query rates as the input for the dynamic lease algorithm. Figure 4.11 illustrates the routine of threshold adjustment.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Nameserver I (b) Nameserver II (c) Nameserver III

Figure 4.13: Storage requirements for given query rates when fixed and dynamic lease are used in DNS traces



(a) Nameserver I (b) Nameserver II (c) Nameserver III

Figure 4.14: Query rates for given storage requirements when fixed and dynamic lease are used in DNS traces

4.4 Performance Evaluation

In this section, we evaluate the effectiveness of dynamic lease of DNScup via trace-driven simulation. Our DNS traces were collected in an academic environment, where three local DNS nameservers provide DNS services for about two thousand client machines. The one-week trace collection is from July 2nd, 2003 to July 9th, 2003. Based on the DNS traces, we simulate a scenario in which a number of clients are using a local DNS nameserver.

Considering the client caching effect on query intervals, we assume that clients cache each resource record for 15 minutes, since this is the default setting in Mozilla. The query rate for each domain name is computed by analyzing the first-day traces. For all three categories of domain names (regular, CDN and Dyn domains), the maximal lease-length is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

set to six days.

4.4.1 Poisson Distribution Validation

Similar to the Web request access patterns, the DNS queries also display bursty behaviors with respect to time series. However, if we force each client to cache the DNS responses, as most Web browsers did, the time interval between two continuous queries for the same record likely follows the Poisson distribution. We use the mean of Coefficient of Variation (CV) to study the query interval distribution in our DNS traces. Figure 4.12 shows the dynamics of the mean of CV with respect to the cache duration at the client side. With the increase of the client cache duration, as the mean of CV is closer to 1, the time intervals are more likely following a Poisson distribution. It is also noticeable that the 95% confidence interval of the mean is very small in all cases.

4.4.2 Experimental Results

We introduce two relative system metrics to evaluate the lease algorithms: storage percentage and request rate percentage. The storage percentage is defined as the ratio between the number of leases granted to requesting DNS caches and the maximal number of leases that an authoritative DNS server could grant. There are two extreme cases: (1) if the authoritative DNS server grants lease to each request and all its RRs have valid leases all the time, the storage percentage is 100%; and (2) if no lease is granted to any request, the storage percentage is 0. The request rate percentage is defined as the ratio between the requests rate issued from a DNS cache and the maximal requests rate that the DNS cache could generate. If no lease is granted, the lease algorithm degrades to the polling scheme

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and generates the maximal request rate. Thus, the request rate percentage becomes 100% under this extreme scenario.

We compare the simple fixed-length lease scheme, which grants the same length lease to every incoming query, with the proposed dynamic lease. Our simulation results clearly demonstrate that the performance of dynamic lease is superior to that of the fixed lease scheme. Figures 4.13 and 4.14 illustrate the simulation results of regular domains based on the traces at three different DNS nameservers. Note that for CDN and Dyn domains, we achieve similar results at all traces.

Dynamic lease is effective to reduce storage overhead. As shown in Figure 4.13 (a), under the request rate percentage of 20%, the storage percentage of dynamic lease is 19% while that of fixed lease is 47%. Dynamic lease reduces storage overhead by 60%. At the same time, dynamic lease is also effective in reducing communication overhead. As shown in Figure 4.14 (a), under the storage percentage of 1%, the request rate percentage of dynamic lease is 56% while that of fixed lease is 88%. The reduction of communication overhead is about 36%.

In our experiments, due to the limitation of the trace length (seven days), the maximal length for regular domains is relatively small. Since regular domains seldom change their DN2IP mappings, we may use a much higher lease-length to gain a better performance. Note that the lease selection in our experiment is done off-line based on the trace analyses, and the lease length remains constant. In reality, a DNS cache may monitor the rates of cached records in the incoming queries. When it detects a significant change in query rates, the DNS cache will notify the authoritative DNS nameserver to re-evaluate the current leases.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5 Prototype Implementation

We build our DNScup prototype on top of BIND 9.2.3. In this section, we first present the extension on the DNS message format to support DNScup mechanism. Then, we describe the structure of the DNScup prototype. Finally, we discuss the security issue related to DNScup. Note that DNScup only keeps cached resource records with valid leases updated, and the rest of the cached resource records still rely on the TTL mechanism to refresh themselves.

4.5.1 Message Formats

In the header of DNS messages, a 1-bit field `QR` is used to specify whether it is a query (0) or a response (1). A 4-bit field `OPCODE` is used to specify the type of the message. In current implementation of BIND, only types 0, 1, 2, 4 and 5 are used and the rest are reserved for future use. To support DNScup, a new opcode 6 in the query/response headers is introduced for lease negotiation. Each DNS query includes the query rate originated from the local clients, and the query rate is expressed in a new 16-bit field `RRC` (recent reference counter) with the domain name being queried at the question section. The authoritative DNS nameserver uses `OPCODE` 6 in the response header to indicate that the lease information is included. If a lease is granted, its duration is specified in a new 16-bit field `TLL` (time of lease-length) at the answer section.

In the BIND 9.2.3 implementation, a message with `OPCODE` of 4 is used for the internal master-slave notification. In order to deal with the wide-area DNS cache update propagation, we define a new type of message called `CACHE-UPDATE`. This message has the same

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ID:	(new)
op:	CACHE–UPDATE(7)
Zone zcount:	1
Zone zname:	(zone name)
Zone zclass:	(zone class)
Zone ztype:	T_SOA

Figure 4.15: The Characteristic Fields of a CACHE-UPDATE Message Header

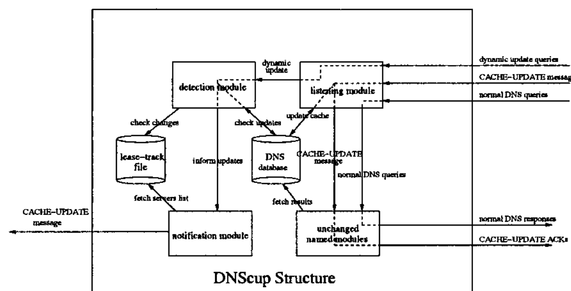


Figure 4.16: The Structure of DNScup Prototype

fields as those in the UPDATE message except for the "op" field in the message header, which is shown in Figure 4.15.

4.5.2 Structure of *DNScup* Prototype

We have modified the prompt notification of the zone mechanism in the BIND 9.2.3 implementation. According to our design, three core components of DNScup have been added to BIND 9.2.3, including the detection module, the listening module, and the notification module. The detection module detects a DNS record change; the listening module monitors incoming DNS queries and updates the track file when necessary; and the notification module propagates DNS CACHE-UPDATE messages. The normal DNS operations remain intact. The interactions among all components are illustrated in Figure 4.16.

For DNS resource records of the authoritative DNS nameserver, the **named** daemon creates a database file to keep track of the incoming DNS queries. Each tuple in this file consists of five fields, which are the source IP address, queried zone name, query type,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

query time, and lease-length. When a DNS query comes in, the `named` first decides if a lease should be granted based on the query rate carried with the query. If yes, a new tuple is added to the track file, and the corresponding response is sent back.

Once a `named` has updated a DNS resource record either annually or via an internal dynamic update message, it retrieves the track file and gets all local DNS nameservers that have queried this record whose leases have not expired yet (i.e., DNS caches). The `named` then sends CACHE-UPDATE messages to these DNS caches through UDP. The notified DNS caches will send back CACHE-UPDATE acknowledgments to the authoritative DNS nameserver and update their cached DNS resource records. The communication process is shown in Figure 4.17.

4.5.3 Secure DNScup

DNScup may impose more concerns on the DNS security. In our current implementation, we transmit DNS messages in plain text for simplicity and efficiency. However, to protect DNS caches against poisoned CACHE-UPDATE messages originated from a compromised DNS nameserver, we need a secure communication channel for cache update. Fortunately,

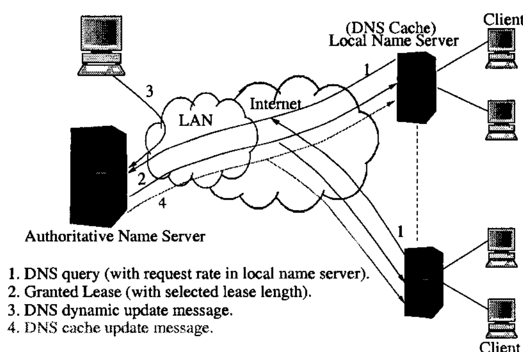


Figure 4.17: DNScup update process.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

DNSSEC [58] and the secure DNS Dynamic Update protocols [122] have been proposed. Coupled with the proposed secure DNS mechanisms, DNScup can achieve a secure cache update without much difficulty.

4.5.4 Preliminary Results

We examine our implementation in a testbed environment — a hierarchy of DNS nameservers in a LAN. The testbed is shown in Figure 4.18. By utilizing multiple virtual IP addresses, we run a master authoritative DNS nameserver and its two slaves on a machine. The root server and two DNS caches are mimicked at three different machines, respectively. The machines used in our experiments are 1GHz Pentium IIIs with 128MB RAM running RedHat Linux 9.1, connected by a 100 Mbps Ethernet. From IRcache [7] proxy traces, we select 50 most popular domain names (46 if excluding "localhost" and three individual IP addresses). A total of 40 zones are constructed for the 46 domain names on the authoritative DNS nameservers, with their glues recorded on the root server. The zone file data are collected through issuing necessary queries to the Internet.

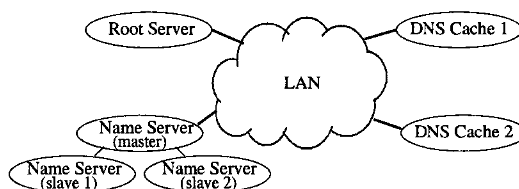


Figure 4.18: DNScup Implementation Testbed

The average lengths of different messages in DNScup are shown in Table 4.2. Compared with existing TTL-based mechanism, the sizes of both query and response messages are increased due to the addition of new fields. However, they are still far below the limitation set by RFC 1035 [87]—a DNS message carried in UDP cannot exceed 512 bytes. Both cache

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

update and its acknowledge messages are small, having sizes similar to those of messages in the DNS dynamic update protocol [103].

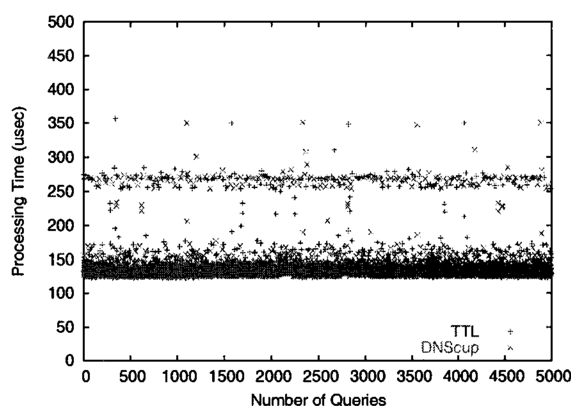


Figure 4.19: DNS nameserver processing overhead: DNScup vs TTL

In order to measure the processing overhead of DNS queries, we set two timers in Bind 9.2.3, one right after receiving a query and the other right before the corresponding response is sent out. The two DNS caches repeat sending queries to the master authoritative DNS nameserver for the 46 collected domain names. After each round, we flush out their cached contents so that the authoritative DNS nameserver can continuously receive and process the queries. Figure 4.19 shows the processing times of 5000 continuous queries with and without DNScup support, respectively. Although DNScup needs to maintain the query rate statistics, the difference in computational overhead between TTL and DNScup is hardly noticeable.

Type	DNScup (Bytes)	TTL (Bytes)	Increment
DNS query	40.8	36.8	10.9%
DNS response	217.8	203.7	6.9%
cache update	80.3	–	–
cache update ack	25.0	–	–

Table 4.2: Average Message Overhead of DNScup

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.6 Summary

In this chapter, we introduced a DNS cache update protocol, called *DNScup*, to maintain strong consistency in DNS caches. To investigate the dynamics of DN2IP mapping changes, we have conducted a wide range of DNS measurements. What we have found is summarized as follows:

- While the physical mapping changes per Web domain name rarely happen, the probability of a physical change per minute within a class is close to one;
- Compared with the frequencies of logical mapping changes, the values of the corresponding TTLs are much smaller, resulting in a large amount of redundant DNS traffic;
- The TTL value of a Web domain name is independent on its popularity, but its logical mapping change frequency is dependent on the popularity of the Web domain.

Based on our measurements, we conclude that maintaining strong cache consistency is essential to prevent potential losses of service availability. Furthermore, with strong cache consistency support, CDNs and other mechanisms can provide fine-grained load-balance, quick responsiveness to network failure or flash crowd, and end-to-end mobility, without degrading the scalability and performance of DNS.

To keep track of the local DNS nameservers whose clients need strong cache consistency for always-on Internet services, *DNScup* uses dynamic lease to reduce the storage overhead and communication overhead. Based on the DNS Dynamic Update protocol, we build a *DNScup* prototype with minor modifications to the current DNS implementation. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

components of *DNScup* implementation include the detecting module, the listening module, the notification module and the lease-track file. Our trace-driven simulation and prototype implementation demonstrate that *DNScup* achieves the strong cache consistency in DNS and significantly improves its availability, performance and scalability.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

P2P Cache Management in Internet

5.1 Introduction

Structured P2P systems have been successfully designed and implemented for global storage utility (such as PAST [106], CFS [51], OceanStore [78], and Pangaea [107]), publishing systems (such as FreeNet [38] and Scribe [28]), and Web-related services (such as Squirrel [68], SFR [117], and Beehive [99]). Among all these P2P-based applications, replication and caching have been widely used to improve scalability and performance. However, little attention has been paid to maintaining replica consistency in structured P2P systems. On one hand, without effective replica consistency maintenance, a P2P system is limited to providing only static or infrequently-updated object sharing. On the other hand, newly-developed classes of P2P applications do need consistency support to deliver frequently-updated contents, such as directory service, online auction, and remote collaboration. In

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

these applications, files are frequently changed, and maintaining consistency among replicas is a must for correctness. Therefore, scalable consistency maintenance is essential to improve service quality of existing P2P applications, and to meet the basic requirement of newly-developed P2P applications.

Existing structured P2P systems rely on distributed hash tables (DHTs) to assign objects to different nodes. Each node is expected to receive roughly the same number of objects, thanks to the load balance achieved by DHTs. However, the system may become unbalanced when objects have different popularities and numbers of replicas. In a scalable replica updating mechanism, the location of a replica must be traceable, and no broadcasting is needed for the propagation of an update notification. Current structured P2P systems take a straightforward approach to track replica locations [113, 102]—a single node stores the locations of all replicas. This approach provides us with a simple solution of maintaining data consistency. However, it only works well if the number of replicas per object is relatively small in a reliable P2P system. Otherwise, several problems may occur as follows.

- *Hot-spot problem*: due to the different objects' popularities, the number of replicas per object varies significantly, making the popular nodes heavily loaded while other nodes carry much less replicas.
- *Node-failure problem*: if the hashed node fails, update notifications have to be propagated by broadcasting.
- *Privacy problem*: the hashed node knows all replicas' locations, which violates the privacy of original content holders.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To address the deficiencies in existing structured P2P systems, we propose a structured P2P system with replica consistency support, called Scalable COnsistency maintenance in structured PEer-to-peer systems (*SCOPE*). Unlike existing structured P2P systems, SCOPE distributes all replicas' location information to a large number of nodes, thus preventing hot-spot and node-failure problems. It also avoids recording explicitly the IP address or node ID of a node that stores a replica, thus protecting the privacy of the node. By building a replica-partition-tree (RPT) for each key, SCOPE keeps track of the location of replicas and then propagates update notifications. We introduce three new operations in SCOPE to maintain consistency.

- *Subscribe*: when a node has an object and needs to keep it up-to-date, it calls *subscribe* to receive a notification of the object update.
- *Unsubscribe*: when a node neither needs a replica nor keeps it up-to-date, it calls *unsubscribe* to stop receiving update notifications.
- *Update*: when a node needs to change the content of an object, it calls *update* to propagate the update notification[1] to all subscribed nodes.

In SCOPE, we allow multiple writers to co-exist, since the update operation on a key can be invoked by any node keeping a replica of that key. In contrast, in some practical applications, usually only one node is authorized to update a key. SCOPE can be easily applied to single-writer applications.

Since SCOPE directly utilizes DHTs to manage object replicas, it effectively supports consistency among a large number of peers. As a general solution, SCOPE can be deployed

[1]invalidation message or the key itself.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in any existing structured P2P systems, such as CAN [102], Chord [113], Pastry [105], and Tapestry [127]. Our theoretical analyses and simulation experiments show that SCOPE can achieve replica consistency in a scalable and efficient manner. In an N-node network, each peer is guaranteed to keep at most $O(\log N)$ partition vectors for a single key, regardless of the key's value and its popularity. Due to the hierarchical management, only $O(1)$ nodes are updated when a node joins or leaves, and only $O(\log^2 N)$ messages are transmitted to recover a node failure.

The remainder of the chapter is organized as follows. Section 5.2 presents the RPT structure in SCOPE. Section 5.3 describes the operations defined in SCOPE. Maintenance and recovery procedures are introduced in Section 5.4. We evaluate the performance of SCOPE using Pastry routing algorithm in Section 5.5. In Section 5.6, we briefly discuss SCOPE design alternatives. Finally, we conclude the paper in Section 5.7.

5.2 The Base of SCOPE Protocol

The SCOPE protocol specifies: (1) how to record the locations of all replicas; (2) how to propagate update notifications to related peers; (3) how to join or leave the system as a peer; and (4) how to recover from a node's failure. This section describes how to record the replica locations by building a *replica-partition-tree* (RPT)—a distributed structure for load balancing in SCOPE. The operation algorithms and maintenance procedures will be presented in Sections 4 and 5, respectively.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2.1 Overview

In DHTs each key is assigned to a node according to its identifier, and we call this original key-holder the *primary node* of the key. To avoid the primary node becoming the hot spot, SCOPE splits the whole identifier space into partitions and selects one representative node in each partition to record the replica locations within that partition. Each partition may be further divided into smaller ones, in which child nodes are selected as the representatives to take charge of the smaller partitions. As the root of this partition-tree, the primary node only records the key existence in the partition one level beneath, while its child representative nodes record the key existence in the partitions two levels below the root; and so on and so forth. In this way, the overhead of maintaining consistency at one node is greatly reduced and undertaken by the representative nodes at lower levels. Since the hash function used by DHTs distributes keys to the whole identifier space, the load of tree maintenance is balanced among all nodes at any partition level. Note that the location information at any level is obtainable from representative nodes at lower levels, the partition-tree also provides a recovery mechanism to handle a node failure.

5.2.2 Partitioning Identifier Space

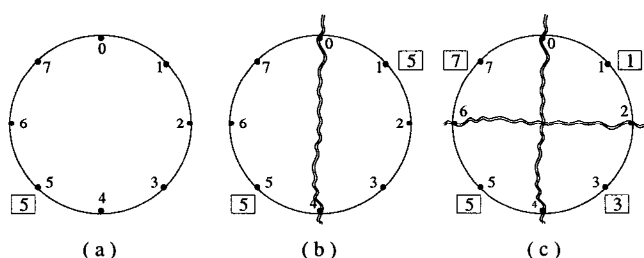


Figure 5.1: (a) A 3-bit identifier space; (b) The same identifier space with two partitions; (c) The same identifier space with four partitions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A consistent hash function (e.g. SHA-1) assigns each node and each key an m-bit identifier, respectively. If we use a smaller identifier space, the key identifier can be easily calculated by keeping a certain number of least significant bits. By adding different most significant bits, the same key can be mapped to multiple smaller equally-sized identifier spaces with different identifier ranges. A partition can be further divided into smaller ones, and it records the existence of all keys in its sub-partitions. Figure 5.1(a) shows an identifier space with $m = 3$. Suppose there is a key 5 in the space. If the original space is split into two partitions as shown in Figure 5.1(b), one with space $[0, 3]$ and the other with space $[4, 7]$, the key can be hashed to 1 in the first partition and 5 in the second partition, respectively. If we further split each partition into two sub-partitions as Figure 5.1(c) illustrated, the identifiers of the same key can be located in the smaller spaces at 1, 3, 5, and 7, respectively. Figure 5.2 shows the root of key 5 (101) in the original 3-bit identifier space and its representative nodes in the two-level partitions. At the top level, the root node is located at 5 (101). At the intermediate level, the two least significant bits (01) are inherited from the root, while the different value (0 or 1) is set at the most significant bit to locate the representative nodes R1 and R2 in the two partitions, respectively. At the bottom level, only the least significant bit (1) is inherited from the root but two most significant bits are set to four different values (00/01/10/11) in order to determine the locations of representative nodes R11, R12, R21, and R22, respectively. Note that the partitioning is logical and the same node can reside in multiple levels. For example, the root node (101) is used as the representative node in all partition levels.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

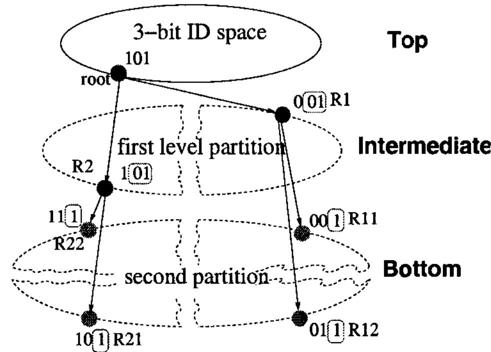


Figure 5.2: Key 5 (101) in a 3-bit identifier space and its representative nodes at different levels of partitions.

5.2.3 Building Replica-Partition-Trees (RPTs)

After partitioning the identifier space as mentioned above, we build an RPT for each key by recursively checking the existence of replicas in the partitions. The primary node of a key in the original identifier space is the root of RPT(s). The representative node of a key in each partition, recording the locations of replicas at the lower levels, becomes one intermediate node of RPT(s). The leaves of RPT(s) are those representative nodes at the bottom level. Each node of RPT(s) uses a vector to record the existence of replicas in its sub-trées, with one bit for each child partition.

Figure 5.3(a) shows an example with the identifier space of $[0, 7]$. The nodes 0, 4, and 7 in the space keep a replica of key 5. The RPT for key 5 is shown in Figure 5.3(b). At the top level, a 2-bit vector is used to indicate the existence of replicas in the two sub-trees. At the bottom level, four 2-bit vectors are used to indicate the existence of key 5 in all eight possible positions from 0 to 7. In general, if the identifier space is 2^M, the height of RPT(s) for any key is $O(M)$. Consider that most DHTs use a 160-bit SHA-1 hashing function, which may result in tens of partition levels. For example, if we split each partition into 64 (2^8) pieces, we will have 20 levels. Obviously, too many levels of partitions would make the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

RPT construction and the update propagation inefficient.

Since the number of nodes is much smaller than the identifier space, our goal is to reduce the heights of RPTs to the logarithm of the number of nodes. In the partitioning algorithm presented above, each partition is recursively divided into smaller ones until only one identifier remains. The leaf nodes of RPTs record the existence of keys at the corresponding identifiers. However, if a partition only contains one node, there is no need for further partitioning to locate the node. For example, as shown in Figure 5.3(a), only node 0 exists in the partition of $[0,3]$. During subscribe/unsubscribe operations, node 0 only needs to inform the primary node of key 5, which records the first level partition $[0,3]$ and $[4,7]$. When the key is modified, it can directly notify node 0 by sending an invalidation message to the first identifier in $[0,3]$, which is 0. By removing the redundant leaf nodes, we can build a much shorter RPT. The RPT after the removal of redundant leaf nodes, is shown in Figure 5.3(c).

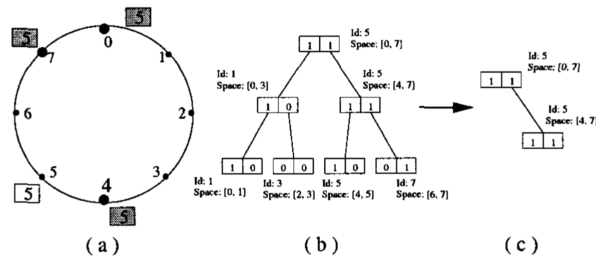


Figure 5.3: (a) In the identifier space of [0,7], nodes 0, 4, and 7 subscribe key 5; (b) The RPT of key 5; (c) The RPT after removing redundant leaf nodes.

The method given above can significantly reduce the partition levels if nodes are distributed sparsely. However, even if the total number of nodes is small, the number of partition levels could still be large when most nodes are close to each other. Figure 5.4(a) shows an example with the identifier space of [0,7], where two nodes 6 and 7 subscribe key

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5. The RPT is illustrated in Figure 5.4(b). Both nodes are in the same partition until the identifier space is decreased to 1—the bottom level of the partition. The height of this RPT is 3, and it cannot be condensed by reducing leaf nodes. In general, if the nodes' identifiers happen to be consecutive and we only remove the leaf nodes as above, the height of RPT(s) will still be $O(M)$.

We resolve this problem by removing the redundant intermediate nodes. If all nodes in a partition are clustered in one of its lower-level partitions, it is possible to reduce the intermediate nodes. Figure 5.4(c) shows one optimized RPT. The intermediate node for the partition $[4, 7]$ is removed since only one of its lower-level partition $[6, 7]$ has nodes. Thus, the height of the RPT is decreased from 3 to 2.

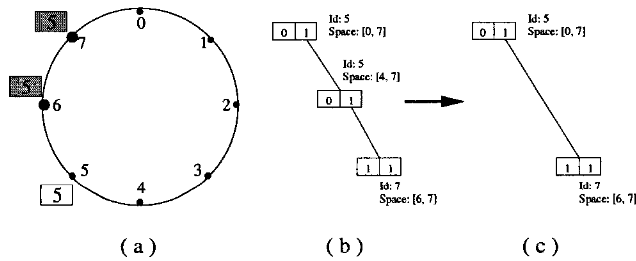


Figure 5.4: (a) Nodes 6 and 7 subscribe key 5; (b) The RPT for key 5 after removing redundant leaf nodes; (c) The RPT after removing both redundant leaf nodes and intermediate nodes.

Theorem 2 *For an N-node network with partition size of 2^m, the average height of RPTs is $O(\frac{\log N}{m})$, regardless of the size of an identifier space.*

Proof: Suppose the whole identifier space is 2^M. Every partitioning generates 2^m smaller equally-sized partitions, each with size of $1/2^m$ of the previous partition range. After $\frac{\log N}{m}$ time partitioning, the identifier range of each partition is reduced to $2^M/2^{\log N} = 2^M/N$. The height of RPT grows to $\frac{\log N}{m}$, with maximal height $\log N$ at $m = 1$. Note that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the expected number of node identifiers in the range of this size is 1. Due to identifier randomness induced by SHA-1 hash function, the average height of all RPTs is $O(\frac{\log N}{m})$. ■

5.2.4 Load Balancing

RPT effectively balances the load across the network, disregarding the key values and their popularities. By using RPT, we conclude that:

Theorem 3 *In an N-node network with partition size of 2^m, for a key with C subscribers, the average number of partition vectors in its RPT is $O(\log N \cdot C)$.*

Proof: In the RPT of the key, only one root is located at the top level. At the second level, at most $\min(2^m, C)$ representative nodes have one partition vector. At the *xth* level of the RPT, at most $\min(2^{xm}, C)$ representative nodes are involved. The total number of vectors S of the RPT is:

$$
\begin{aligned}
S &= 1 + \min(2^m, C) + \min(2^{2m}, C) \\
&\quad + \ldots + \min(2^{\frac{\log N}{m} m}, C) \\
&= \sum_{i=0}^{a-1} 2^{im} + \sum_{i=a}^{\frac{\log N}{m}} C \\
&= (\log_m N - a)C + \frac{2^{am} - 1}{2^m - 1}, \\
&\quad \text{for } 2^{(a-1)m} < C \leq 2^{am}, 1 < a \leq \tfrac{\log N}{m}
\end{aligned}
$$

Compared with the number of subscribers C, the number of vectors is increased to $(\log N - a) + \frac{2^{am}-1}{(2^m-1)C}$. Since $\frac{2^{am}-1}{(2^m-1)C} < \frac{2^{am}}{(2^m-1)2^{(a-1)m}} = \frac{2^m}{2^m-1}$, which is less than 2 ($m \geq 1$), the maximal value is achieved when $a = 1$, and the total number of vectors in the RPT is $O(\log N)$ times of the number of subscribers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3 Operation Algorithms

5.3.1 Subscribe/Unsubscribe

The subscribe/unsubscribe procedures are initiated by subscribers and proceed toward the root of an RPT. The process can be implemented in an iterative or recursive way. With iteration, the subscriber itself has to inform all representative nodes one by one. With recursion, each representative node is responsible for forwarding the subscriptions to the next higher level until the root node is reached. In SCOPE, we implement the subscribe/unsubscribe operations recursively for routing efficiency.

At the beginning, each subscriber locates its immediate upper-level partition from its predecessor's and successor's identifiers. Then, the node sends subscribe/unsubscribe requests to the upper-level representative node. The representative node checks if it has a vector allocated for the key. If so, it sets/unsets the corresponding bit, and the subscribe/unsubscribe procedure terminates there. Otherwise, it creates/deletes the vector of the key, sets/unsets the bit, and continues forwarding subscribe/unsubscribe requests to the representative node at the next upper-level partition. This process proceeds until it reaches the root of the RPT. The routing algorithms of the operations depend on the type of the specific structured P2P systems. In this section, we use Pastry as the base routing scheme for the purpose of analysis. Note that similar analysis is applicable to other hypercube routing algorithms as well.

Figure 5.5 illustrates a subscribe/unsubscribe process in a 3-bit identifier space, where

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

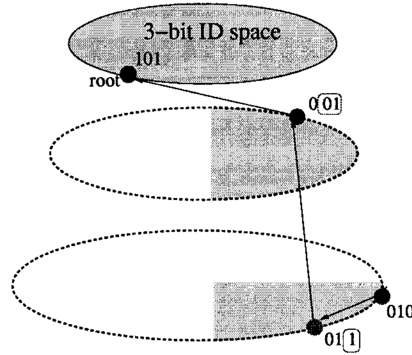


Figure 5.5: Node 2 (010) subscribe Key 5 (101) in a 3-bit identifier space.

node 2 (010) subscribes/unsubscribes key 5 (101). At first, node 2 notifies the representative node 3 (011) at the bottom level, then node 3 informs the representative node 1 (001) at the intermediate level. Finally, node 1 informs the root node 5, which is the representative node of the whole space.

In order to improve routing efficency, every node maintains level indices to indicate the node's partitions at different levels. As we have pointed out before, reducing intermediate partitions makes the height of RPT different from the depth of partitioning. The level index is used to record the change of the RPT height with the increase of partitioning. Its maximal length is equal to M for an identifier space with size of 2^M. The ith entry in a level index is the height of the RPT at i^{th} partition level.

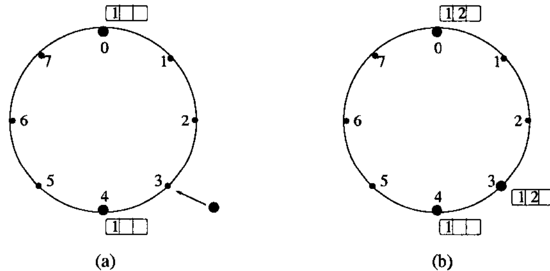


Figure 5.6: Level index changes after node 3 joins in a 3-bit space.

Figure 5.6 shows an example of the level index in a 3-bit identifier space. Before node

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3 joins, nodes 1 and 4 are identified after first-time partitioning. Both of them have the same level index $\{1, \sqcup, \sqcup\}$, where $\sqcup$ represents an empty space. With the participation of node 3, the whole space needs to be partitioned twice to identify nodes 0 and 3, and no redundant intermediate partition exists. The RPT grows as partitioning proceeds, and the level indices of nodes 0 and 3 become $\{1, 2, \sqcup\}$. Comparatively, node 4 is identified after the first-time partitioning and its level index is $\{1, \sqcup, \sqcup\}$.

With the Pastry routing table and leaf set, a node can reach any other node in a range of 2^{xm} within $O(\log(\frac{N}{2^{M-xm}}))$ hops. When a node initiates a subscribe/unsubscribe operation, it also forwards its level index to the representative nodes at upper levels. Each intermediate representative uses the level index to derive the location of its higher level representative.

Lemma 1 *For an N-node network with partition size of 2^m in a 2^M identifier space, in any range of 2^{xm}, on average a node can find the successor of a key in $O(\log(\frac{N}{2^{M-xm}}))$ hops.*

Proof: We use Pastry as the base routing algorithm, and assume that a node's routing table is organized into $\lceil log_{2^b} M \rceil$ levels with $2^b - 1$ entries each. Due to the usage of the SHA-1 hash function, node and key identifiers are randomly distributed and the number of nodes in a range of 2^{xm} is $\frac{N}{2^{M-xm}}$ on average. Suppose that node n wants to resolve a query for the successor of k, $k > n$ and $0 < k - n < 2^{xm}$. Node n forwards its query to the close predecessor of k in its routing table. Suppose key k is in the ith $(0 < i <= xm/b)$ level j^{th} $(0 < j < 2^b)$ interval of node n. If this interval is empty, node n has found the successor of k. Otherwise, it fingers some node f in this interval. The distance between n and f is at least $2^{xm-i\cdot b}$. Since f and k are both in i's level, the distance between them is at most $2^{xm-i\cdot b}$, less than half of the distance from n to k.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

After $\log(\frac{N}{2^{M-xm}})$ forwardings, the distance between the current query node n and key k is reduced to $2^{xm}/2^{\log(\frac{N}{2^{M-xm}})} = 2^M/N$. Because the expected number of node identifiers in this range is 1, the successor of key k can be reached in $O(\log(\frac{N}{2^{M-xm}}))$ hops on average. If $n > k$ and $0 < n - k < 2^{xm}$, we can obtain the same result. ■

Theorem 4 *For an N-node network with partition size of 2^m, on average the subscribe/unsubscribe operations can be finished in $O(\frac{\log^2 N}{2m})$ hops.*

As we have learned from the previous section, the average height of RPT(s) is $O(\frac{\log N}{m})$. In order to finish a subscribe/unsubscribe operation, $O(\frac{\log N}{m})$ levels are traversed on average. From Lemma 1, at each level, on average a query node can reach the successor of a key in the same partition at level l in $\log N/2^{lm}$ hops, the average routing length of a subscribe/unsubscribe operation ($hop(sub/unsub)$) is:

$$\begin{aligned} hop(sub/unsub) &= \log N + \log \frac{N}{2^m} + \log \frac{N}{2^{2m}} \\ &\quad + ... + \log \frac{N}{2^{\log N}} \\ &= \log N + (\log N - m) + (\log N - 2m) \\ &\quad + ... + (\log N - \frac{\log N}{m} m) \\ &= \frac{\log^2 N}{2m} - \frac{\log N}{2}. \end{aligned}$$

Therefore, on average a subscribe/unsubscribe operation can be finished in $O(\frac{\log^2 N}{2m})$ hops.

While the upper bound of subscribe path length is longer than that of the centralized solution, the subscribe/unsubscribe operations in SCOPE are efficient if multiple subscribers exist. The subscribe/unsubscribe process terminates at any representative node that has

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

recorded the same key at the same level. When a node subscribes a key with C replicas, the average length of the subscribe/unsubscribe process can be expressed as follows:

$$\begin{aligned} hop(sub/unsub) &= \log N + \log \frac{N}{2^m} + ... + \log \frac{N}{2^{\log N}} \\ &\quad -(\log N + \log \frac{N}{2^m} + ... + \log \frac{N}{2^{\log C}}) \\ &= \sum_{i=0}^{\frac{\log N}{m}} \log \frac{N}{2^{im}} - \sum_{i=0}^{\frac{\log C}{m}} \log \frac{N}{2^{im}} \\ &= \sum_{i=\frac{\log C}{m}+1}^{\frac{\log N}{m}} \log(N/2^{im}) \end{aligned}$$

5.3.2 Update

The update procedure is launched by the root node after it receives update requests from a replica, and proceeds toward every subscriber. The root node first checks its vector of the key. Then, it sends notifications to the representative nodes of the partitions with corresponding bits set. Every intermediate representative is responsible for delivering the notifications to its lower-level representatives. When the notification reaches a leaf node, it is forwarded to the subscribers directly.

Theorem 5 *For an N-node network with partition size of 2^m, on average, update operations can be finished in $O(\frac{\log^2 N}{2m})$ hops.*

Although the update path to a single subscriber in SCOPE is longer than that in the centralized solution, the average number of update routing hops in SCOPE is smaller. This is because in SCOPE all replicas can be notified in $O(\log^2 N)$ hops, but the centralized

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

solution needs $O(C)$ hops to finish. For a sufficiently large C, the latter incurs much longer delay.

Similar to subscribe/unsubscribe operations, the update operation in SCOPE is efficient if multiple subscribers exist. The total number of hops (*hop*(*update*)) for an update of a key with C replicas is:

$$\begin{aligned} hop(update) &= C\log N - (C-2^m)\log N + C\log\frac{N}{2^m} \\ &\quad -2^m(\frac{C}{2^m}-2^m)\log\frac{N}{2^m} + ... + C\log\frac{N}{2^{\frac{\log C}{m}m}} \\ &\quad -2^{\frac{\log C}{m}m}(C/2^{\frac{\log C}{m}m}-2^m)\log\frac{N}{2^{\frac{\log C}{m}m}} \\ &\quad +C\log\frac{N}{2^{(\frac{\log C}{m}+1)m}} + ... + C\log\frac{N}{2^{\frac{\log N}{m}m}} \\ &= C\sum_{i=\frac{\log C}{m}+1}^{\frac{\log N}{m}}\log\frac{N}{2^{im}} + \sum_{i=0}^{\frac{\log C}{m}}2^{(i+1)m}\log\frac{N}{2^{im}} \end{aligned}$$

Comparatively, the centralized solution needs $O(C\log N)$ hops to conduct an update operation. Our analysis above is based on the base SCOPE protocol, in which we do not record the IP addresses of descendant nodes explicitly. To further reduce the latency of update operations, a parent node may directly record the IP addresses of its RPT children. If so, SCOPE can reduce the average update hops to $O(\log N)$, which is much smaller than that of the centralized solution for a sufficiently large C.

5.4 Maintenance and Recovery

5.4.1 Node Joining/Leaving

This section describes how to maintain the RPT when a single node joins. A similar method can be applied to the situation where a node leaves.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

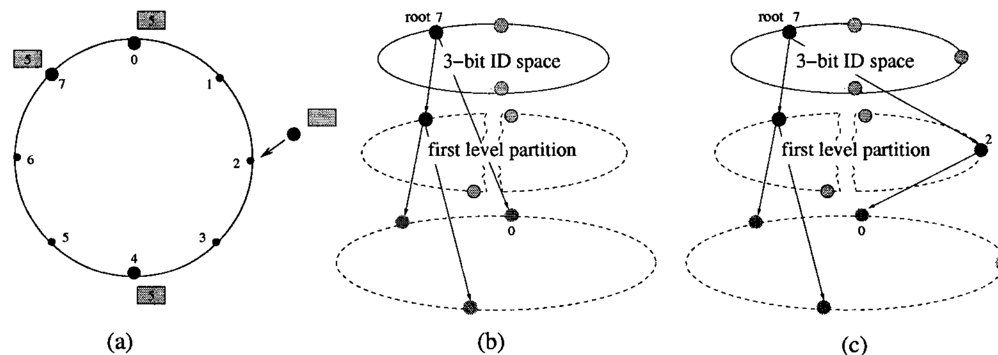


Figure 5.7: (a) Node 2 (010) joins in a 3-bit identifier space; (b) The RPT of key 5 before node 2 joins; (c) The RPT of key 5 after node 2 joins.

Besides maintaining the predecessor/successor and routing tables, a newly-joining node in SCOPE needs to take two actions to maintain the RPT : transferring partition vectors and updating level indices. With a node joining, part of RPT vectors under the charge of the node's successor should be transferred to the newly-joined node, similar to transferring keys. The operation is straightforward: the newly-joined node n informs its successor n', then node n' moves the related vectors at every level to node n.

A node joining may cause further partitioning to distinguish itself from other existing nodes. If node n changes the structure of any RPTs, it also needs to inform the affected nodes to update their level indices accordingly. Figure 5.7 illustrates the level index maintenance when node 2 joins a 3-bit identifier space. Figure 5.7(a) shows the node distribution. Originally, only node 0 is in the partition $[0, 3]$, and its level index is $\{1, \sqcup, \sqcup\}$. Figure 5.7(b) shows the RPT of key 5, where node 7, the primary node of key 5, can read the first bit of its vector and know the existence of key 5's replica at node 0. After node 2 joins, node 0 is no longer the only node in the partition $[0, 3]$, thus further partitioning within [0, 3] becomes necessary to differentiate node 0 from node 2. Subsequently, node 0 updates its level index to $\{1, 2, \sqcup\}$ and subscribes key 5 via node 2—the representative of the new partition. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

modified RPT of key 5 is illustrated in Figure 5.7(c). Note that a newly-joined node triggers at most one partitioning. The newly-generated partition consists of the newly-joined node and at least one existing node.

Lemma 2 *In an N-node network, when a node joins, on average only $O(1)$ nodes need to update their level indices.*

Proof: According to the design of RPTs, we do not need to partition at a level if only one of its lower partitions has nodes. If the newly-joined node makes the partitioning a necessity, only the existing nodes in that lower partition need to update their level indices. Considering that nodes are randomly distributed among partitions, the average number of nodes in that partition is $O(1)$. ■

Theorem 6 *In an N-node network with partition size of 2^m, on average, any node joining or leaving requires $O(1)$ messages to update the corresponding RPTs and level indices.*

When a node joins/leaves, both RPTs and level indices can be updated with $O(1)$ messages. The total number of maintenance messages is still $O(1)$.

5.4.2 Node Failure

One advantage of multi-level partitioning is fault tolerance. The records at any level partition can be restored from its lower-level partitions. SCOPE has a recovery process invoked periodically after the stabilization process. When one peer fails, the recovery process is executed by the new node that takes over the failed one. The new node sends queries to its lower-level partitions, then restores every key's vectors based on their responses.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Suppose a node fails in an N-node network with partition size of 2^m. In order to recover the RPTs, another node taking over the failed node needs to collect all subscription information from all 2^m lower-level partitions at all $\frac{\log N}{m}$ levels. On average, the total number of messages ($Message$) is:

$$\begin{aligned} Message &= 2^m(\log N + \log \frac{N}{2^m} + \log \frac{N}{2^{2m}} \\ &\quad +... + \log \frac{N}{2^{\log N}}) \\ &= 2^m \frac{\log^2 N}{2m} - 2^m \frac{\log N}{2} \end{aligned}$$

Thus, when a node fails, the recovery process only needs $O(2^m \frac{\log^2 N}{2m})$ messages.

Comparatively, a centralized approach can recover the replica locations by broadcasting the re-subscription requests to all nodes or by requiring all subscribed nodes to periodically communicate with the hashed nodes. Obviously, neither of these methods is as effective as the method that SCOPE uses.

5.5 Performance Evaluation

In this section, we validate the efficacy of SCOPE through simulations. In our experiments, all nodes and keys are randomly-selected integers. They are hashed to a 160-bit identifier space via SHA-1. The number of partitions at each level is 16. Specified as the Pastry default parameters, the routing table of each node has 40 levels and each level consists of 15 entries; the leaf set of each node has 32 entries.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

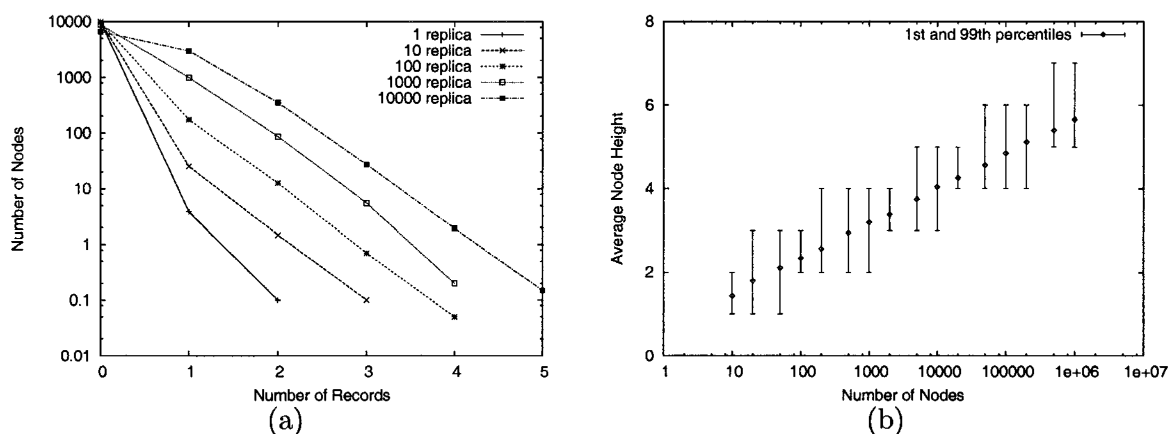


Figure 5.8: (a) The storage distribution of different nodes; (b) The average RPT height with the change of number of nodes.

5.5.1 Structure Scalability

A scalable P2P system should distribute the whole storage load to a large number of nodes to avoid the hot-spot problem. We consider a network consisting of 10^4 nodes, and vary the total number of replicas of a key at 1, 10, 10^2, 10^3 and 10^4. In order to record a key and its locations, a record [*key_id, partition_level, partition_vector*] is kept at each representative node. We measure the number of nodes involved and the number of records stored on each node to evaluate scalability. The experiments are repeated 20 times and the mean values are plotted in Figure 5.8(a). With the increase of total replicas, the number of records on a node is slowly increased. For example, when there are 10^4 replicas, it is rare for a node to have more than three records.

The height of an RPT determines the latency of operations in SCOPE. By varying the number of nodes, we measure the level of partitions for each node, which is equal to the height of RPTs of all replicas on one node. Figure 5.8(b) plots the mean, the 1st, and 99th

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

percentiles of the height of RPTs with the increase of number of nodes. The RPT heights of all keys residing in different nodes exhibit small variations, and they grow logarithmically with the increase of the number of nodes.

Assume subscribers follow a Zipf's distribution, and there are 10^4 and 10^5 keys in a 10^4-node network. The number of subscribers of ith most popular key is equal to $1/i$ of total number of nodes. Figures 5.9(a) and (b) plot the storage load on each node when the number of total keys is 10^4 and 10^5, respectively. The storage load is measured by the number of records kept on one node. Compared with the centralized solution, SCOPE can effectively distribute the storage load to all nodes, thus avoiding the hot-spot problem. In these two experiments, the maximal records on a single node are reduced from 10004 and 10078 in the centralized solution to 105 (1/95) and 387 (1/26) in SCOPE, respectively.

Next we consider a query distribution obtained from Web proxy logs [7]. We randomly selected 10^4 and 10^5 hostnames, and the distribution of the number of subscribers is equal to that of requests collected during one week period (Nov. 02 - Nov. 08, 2003). Figures 5.9(c) and (d) plot the storage load on each node when the number of total keys is 10^4 and 10^5, respectively. Again, the maximal number of records on a single node are reduced to 418 and 1937 in SCOPE, which are 24 and 10 times less than 10098 and 19548 in the centralized solution, respectively.

5.5.2 Operation Effectiveness

In this section, we evaluate the effectiveness of new operations in SCOPE. We focus on subscribe operations only, since the other two kinds of operations (unsubscribe/update) are similar to subscribe operations. Figure 5.10(a) plots the dynamics of the subscribe path

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

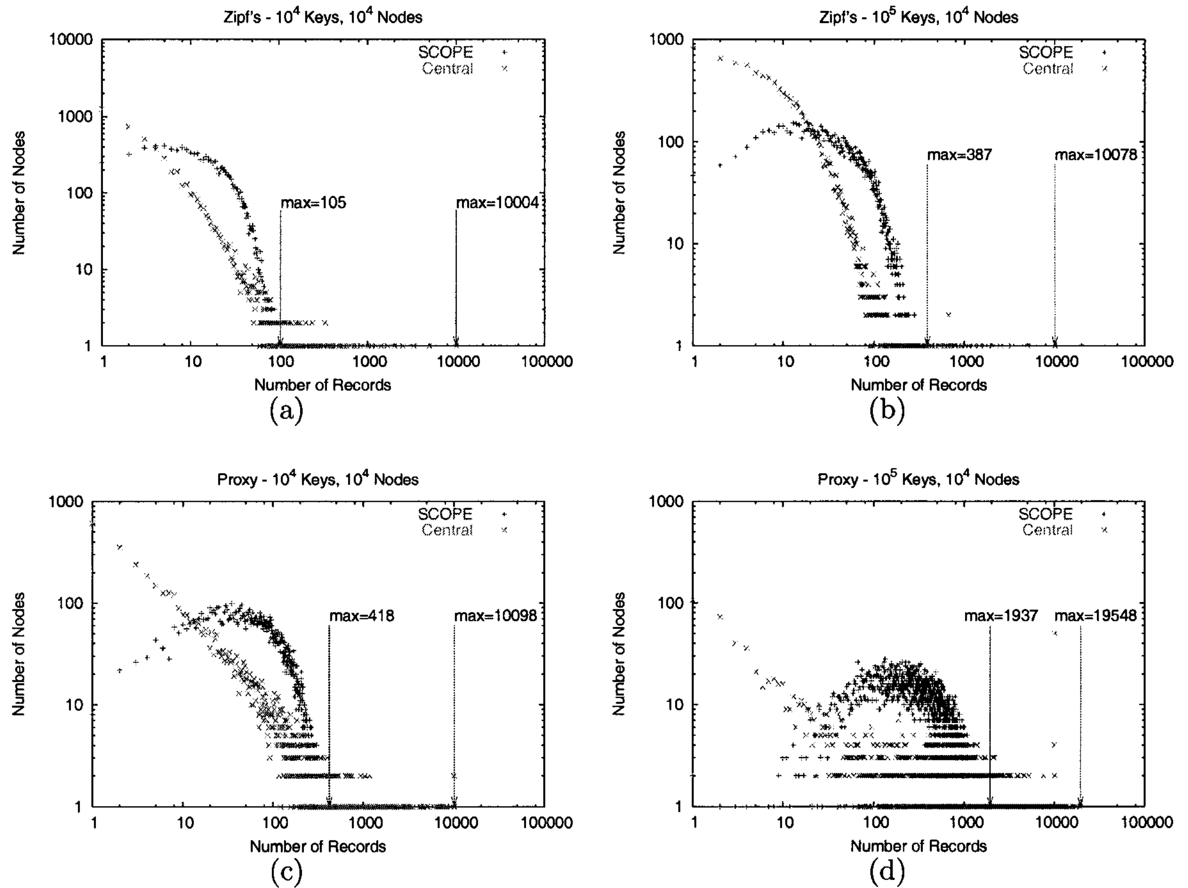


Figure 5.9: With the Zipf's distribution, the number of records kept by each node in: (a) 10^4-key, 10^4-node network; (b) 10^5-key, 10^4-node network; with the distribution obtained from proxy logs, the number of records kept by each node in: (c) 10^4-key, 10^4-node network; (d) 10^5-key, 10^4-node network.

length with the increase of total number of nodes. SCOPE has longer paths to finish a subscribe operation than the centralized solution, because multiple representative nodes should be contacted before the primary nodes are reached. In practice, when numerous subscribers exist, the subscribe operation may terminate at a representative node, leading to a reduced path. Figure 5.10(b) illustrates the effects of multiple subscribers on the path length. If the number of subscribers is larger than 200 in a 10^4-node network, the path

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

length is shorter than the centralized solution.

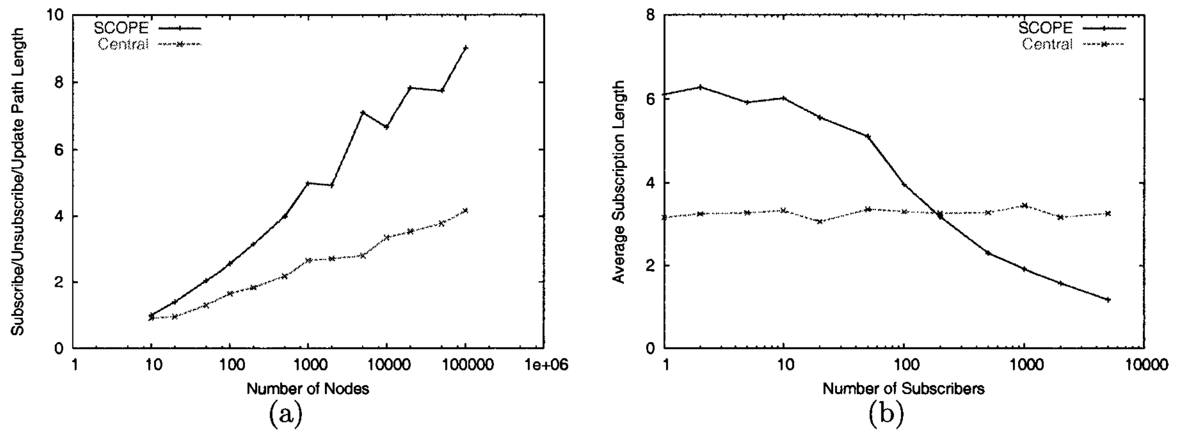


Figure 5.10: (a) The subscribe operation path length in SCOPE compared with the centralized solution; (b) The changes of subscribe operation path length with the variance of number of subscribers.

Assume subscribers follow a Zipf's distribution. Figures 5.11(a) and (b) plot the distribution of the messages sent/received by each node in a 10^4-node network with 10^4 and 10^5 keys, respectively. Although the maximal path length in SCOPE is longer than that of the centralized solution, the messages from the subscribe operations are much more evenly distributed among all nodes, instead of clogging at a few nodes. As the simulation results shown, when the number of keys is 10^4, the maximal number of messages on a single node is 412 in SCOPE, only about 1/25 of 10445 in the centralized solution. When the number of keys is 10^5, the maximal number of messages on a single node increases to 1410 in SCOPE, but still only about one seventh of 10406 in the centralized solution.

When the subscriber distribution is obtained from proxy logs [7], Figures 5.11(c) and (d) illustrate the distribution of the messages sent/received by each node in a 10^4-node network with 10^4 and 10^5 keys, respectively. The maximal number of messages on a single

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

node in SCOPE is only one sixth (1906 vs. 11218) and one fourth (7925 vs. 33068) of the centralized solution in these two cases, respectively.

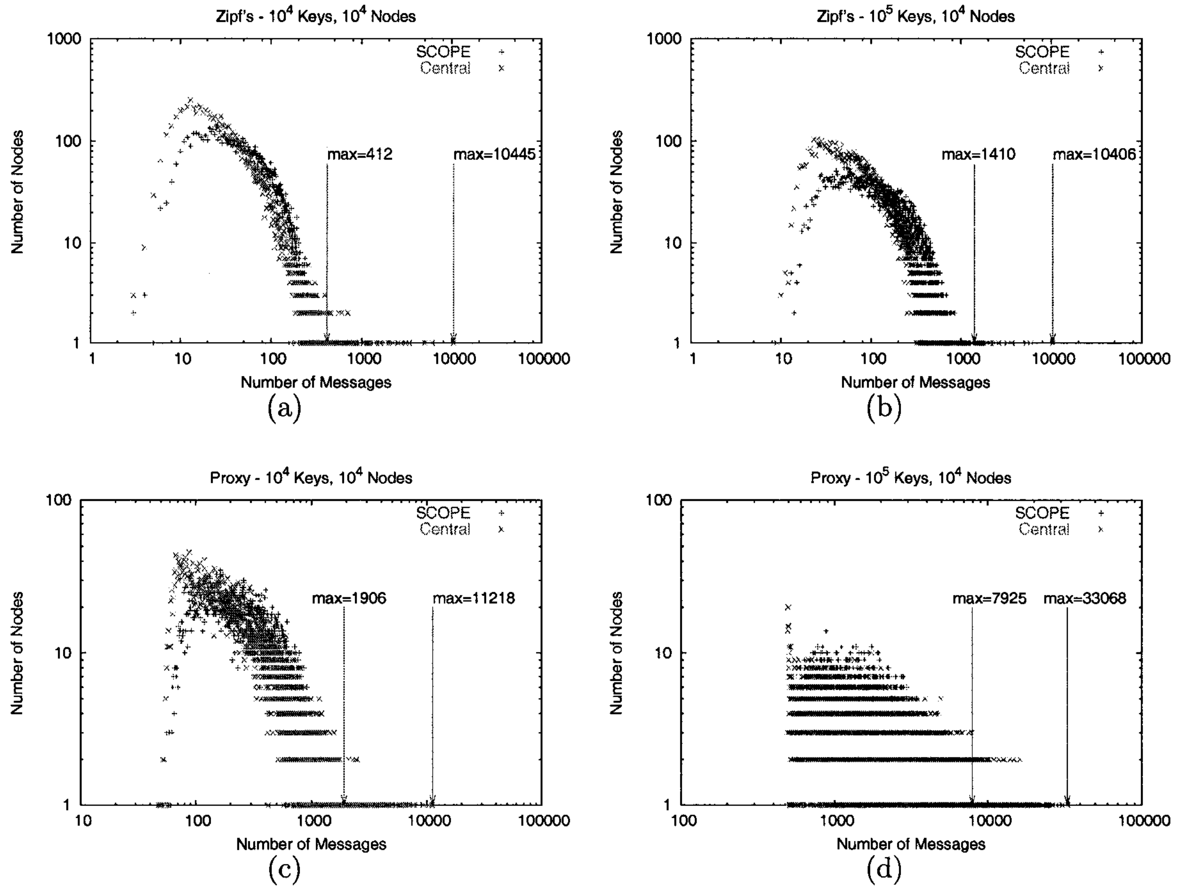


Figure 5.11: With the Zipf's distribution, the distribution of messages sent/received by each node in: (a) 10^4-key, 10^4-node network; (b) 10^5-key, 10^4-node network; with the distribution obtained from proxy logs, the distribution of messages sent/received by each node in: (c) 10^4-key, 10^4-node network; (d) 10^5-key, 10^4-node network.

5.5.3 Maintenance Cost

The maintenance cost includes node joining and leaving, and node failure recovery. Besides the maintenance routines in Pastry, node joining/leaving needs additional operations to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

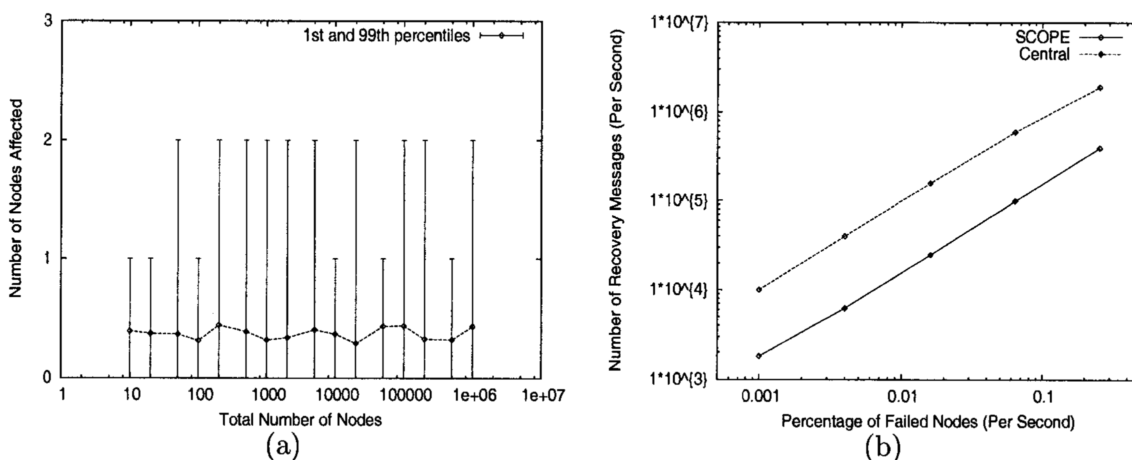


Figure 5.12: (a) The number of nodes to update their level indices at a node joining; (b) The number of maintenance messages in a network with certain node failure rate.

maintain an RPT. We focus on the additional overhead induced by SCOPE. The first part of maintenance, *transferring RPT*, can be completed through a regular operation in Pastry. The second part of maintenance, *further partitioning*, needs additional operations. Figure 5.12(a) illustrates the number of affected nodes when a new node joins. On average, a newly-joined node only invokes 0.5 node to update its level index, disregarding the size of a P2P system.

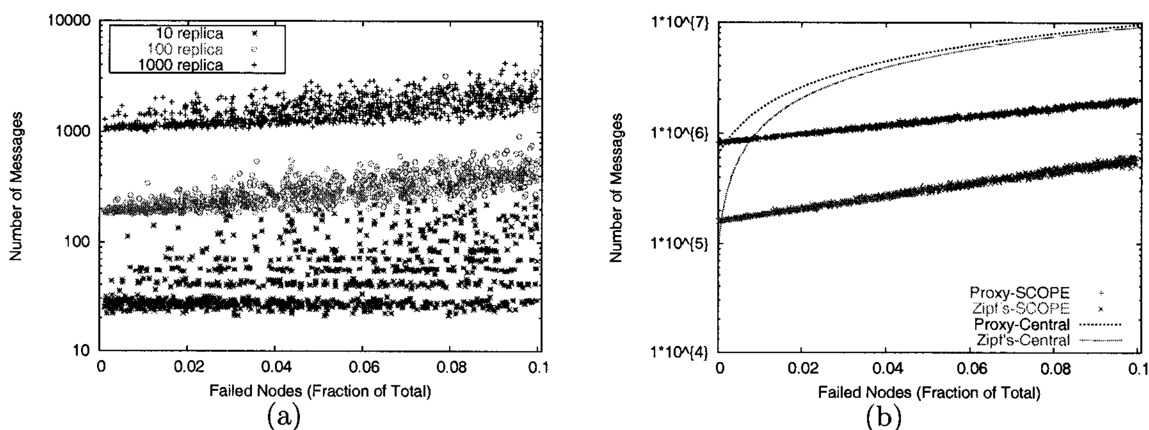


Figure 5.13: (a) The number of messages for an update in a 10^4-node network; (b) The total messages for updates in a 10^4-key, 10^4-node network for both Zipf's and proxy log-based distributions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

One important feature in SCOPE is its efficient recovery mechanism when a node failure is detected. As in Pastry, in order to detect node failures, the neighboring nodes periodically exchange keep-alive messages. In our experiments, we only count the additional messages in the network to recover from node failures. In SCOPE, the new representative nodes communicate with the lower-level representatives to recover RPTs in case of a node failure. On the contrary, the centralized solution has to broadcast the recovery information to all nodes. Figure 5.12(b) shows the message rate in a 10^4-node network with given node failure rates. Compared with the centralized solution, SCOPE only consumes about one fifth of messages to recover under different failure rates.

5.5.4 Fault Tolerance

When a node fails, in order to propagate the update, a centralized scheme relies on broadcasting to reach the destination nodes. In contrast, SCOPE only needs to send the update notifications to representative nodes at the lower levels. We simulate a network with 10^4 nodes and 10^4 keys. Figure 5.13(a) plots the total number of messages of an update operation with the increase of failed nodes. Note that the update is made on an object with 10, 100, or 1000 replicas, respectively. In all cases, the number of messages for the update is proportionally increased with the fraction of failed nodes. Assume subscribers follow either Zipf's or proxy log-based distribution, Figure 5.13(b) plots the total number of messages if all keys are updated once. The number of messages in SCOPE is about 5%-10% of the message overhead in the centralized scheme for both Zipf's and proxy log-based distributions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.6 Design Alternatives

Selected Representative Nodes

Frequent node joins and leaves, which is not uncommon in practice [20], could significantly degrade the performance of SCOPE. However, we can mitigate this performance degradation by pre-selecting representative nodes in RPTs. Not all nodes are eligible for being representative nodes; only those trusted and stable ones are selected as representatives for each partition. Since transient nodes join at the bottom level, the higher-level partitions are relatively stable and the cost of maintaining RPTs is minimized.

Direct Notification

In the base SCOPE protocol, the update process needs to traverse multiple partitions even if only one replica remains in the P2P system. If privacy is not a concern, SCOPE can be easily extended to record subscribers' IP addresses to shorten the latency of update notifications. If a node/partition is the first one to subscribe a key in the upper-level partition, this partition records its IP address in addition to the partition vector. When the upper level receives an update notification, it directly forwards the message to the node with the corresponding IP address without traversing the partition tree.

Dynamic Partitioning

The partition number at each level is predefined and fixed in the base SCOPE. A large number of partitions may reduce the total number of levels, thus lower subscribe/unsubscribe/update latency, but at the expense of high space overhead caused by a large number of partition vectors. On the other hand, a small number of partitions may increase the number of levels

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with low space overhead. Considering the tradeoff between routing latency and storage overhead, our partitioning scheme could be dynamic, in which the number of partitions is adaptively changed with respect to the popularity of a key. In this scheme, the root of the RPT for a key decides the appropriate number of partitions for that key. Since subscribers of the key do not know the number of partitions, subscribe/unsubscribe operations always start from the root RPT vectors.

5.7 Summary

The challenges to building a consistent P2P system are twofold: large scale and high failure rates. In this chapter, we proposed a scalable, consistent structured P2P system, called SCOPE. Based on structured DHTs, SCOPE builds a replica-partition-tree for each key to distribute its replica location information among peers. In an N-node network, each peer is guaranteed to keep at most $O(\log N)$ partition vectors for a single key, regardless of the key's value and its popularity. Three new primitives, subscribe/unsubscribe/update, are introduced specifically to maintain the replica consistency. Due to hierarchical management, these operations can be committed efficiently with minimal maintenance cost. Only $O(1)$ nodes are updated when a node joins or leaves, and only $O(\log^2 N)$ messages are transmitted to recover a node failure. Our theoretical analyses and simulation experiments have shown that SCOPE scales well with the number of nodes, maintains consistency effectively, and recovers from node failures efficiently.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6

Conclusion and Future Work

The objective of this dissertation is to address the limits of existing Internet latency reduction techniques, including Web prefetching, DNS cache consistency and P2P cache management. This dissertation introduces a set of new efficient solutions to significantly improve the efficiency and performance.

6.1 Existing Problems

As general solutions to reduce latency, caching, replication and prefetching have been deployed in Internet to improve the client perceived response time. This dissertation is motivated by the following limitations found in current Internet:

- Web prefetching decisions are often made through pre-defining probability thresholds. It can limit the generated overheads on both Web servers and networks by selecting the prefetched objects with access probabilities larger than the thresholds. The thresholds also determine the tradeoffs between the improved client perceived latency. The effectiveness of this static scheme depends on the assumption that more

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

aggressive prefetching makes lower latency with higher overhead consumption, which is not ture in practice. When either server or network is over loaded, prefetching may increase the latency due to the processing delay.

- As a major component in Internet, DNS only has a simple TTL-based scheme to refresh the DNS cache content. Initially designed around 20 years ago, it worked efficiently in the past when the mappings between the domain names and IPs were rarely changed. However, in most recent years, the mappings became dynamic because of the deployments of new techniques such as CDN and dynamic IPs. The original TTL-based solution cannot effectively maintain the cache consistency. Thus the loss of connections is inevitable for the domains with dynamic mappings.

- P2P systems use replication to improve the system scalability and dependability. Although P2P systems are designed to scale to millions of nodes, existing cache management schemes are limited either in scalability or in reliability. Since P2P networks are mostly used by sharing static objects, existing schemes are acceptable even with the potential problems. But the lack of effective cache management prevents P2P networks from serving the applications for dynamic objects.

6.2 Contributions

The major theme of the dissertation is reducing Internet latency by utilizing rich bandwidth and large storage capacity. The contributions of the dissertation are concisely summarized in table 6.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 6.1: Dissertation Contributions

Base Techniques	Where in Internet	Identified Problems	Objectives/accomplishment
Caching	DNS	weak consistency	strong consistency
Prefetching	Web	low efficiency	algorithmic and system improvement
Replication	P2P	expensive consistency maintenance	scalable consistency maintenance

1. **DNScup: a new DNS cache consistency maintenance scheme based on active update propagation.** To keep tracks of local DNS nameservers whose clients need strong cache consistency for always-on Internet services, DNScup uses dynamic lease algorithm to balance the storage overhead and the communication overhead. Based on the DNS Dynamic Update Protocol, we design and build a DNS Cache Update Protocol (DNScup) prototype with minor modifications to the current DNS implementation. Our trace-driven simulation and prototype implementation demonstrate that DNScup achieves the strong cache consistency in DNS and significantly improves its availability, performance and scalability.

2. **A measurement-based coordinated Web prefetching framework for supporting predictable response time.** This framework allows both Web servers and proxies provide Web prefetching supports in a coordinated and dynamic way. Each of them utilizes the its local Web access information to make predictions. Web servers will make predictions only when the proxies cannot provide prediction results for incoming requests from clients. Thus, proxies can reduce the computation overhead on Web servers and the communication overhead on Internet while the prediction accuracy is improved. Based on measuring server and network utilization,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

both Web server and proxies also decide the prefetching aggressiveness by adjusting the prefetching thresholds dynamically. The adjustment process is performed periodically to minimize the client perceived latency at real time.

3. **SCOPE: a scalable DHT-based P2P cache management scheme.** Based on existing Distributed Hash Tables (DHTs), SCOPE builds a replica-partition-tree for each key to distribute its replica location information among peers. Replica partition tree (RPT) of every key is constructed by aggregating the information with good scalability. In an N-node network, each peer is guaranteed to keep at most $O(\log N)$ partition vectors for a single key, regardless of the key's value and its popularity. Three new primitives, subscribe/unsubscribe/update, are introduced specifically to maintain the replica consistency. Due to hierarchical management, these operations can be committed efficiently with minimal maintenance cost. Only $O(1)$ nodes are updated when a node joins or leaves, and only $O(\log^2 N)$ messages are transmitted to recover a node failure. Our theoretical analyses and simulation experiments have shown that SCOPE scales well with the number of nodes, maintains consistency effectively, and recovers from node failures efficiently.

6.3 Future Work

There are three general areas of future work suggested by our research. First, we have evaluated the measurement-based coordinated Web prefetching framework on a LAN. A logical extension would be to construct a cluster of servers and perform the experiments on the real Internet. One issue that requires further research would be the adjustment

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

accuracy when multiple kinds of services are deployed on servers simultaneously.

Second, we validated DNScup by building a simple DNS system on limited number of nodes. Although DNScup is scalable in nature, its performance and reliability should be examed in large scale systems for real deployment in Internet. Since DNScup changes the communication procedures among name servers, the existing security mechanism should be changed accordingly to guarantee the authority of the propagation from remote name servers.

Finally, SCOPE provides a general way to manage replications in P2P cache while many applications have specific requirements. Specifically, when multiple writers exist, more strict rules should be enforced to make all peer readers have the same responses. Another related issue worthing further investigation is to construct adaptive P2P systems based on their scales for lower maintenance overhead and higher reliability.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

[1] Akamai technologies, inc. http://www.akamail.com/.

[2] Apache http server project. http://httpd.apache.org/.

[3] Content delivery and distribution networks. http://www.web-caching.com/cdns.html.

[4] Dynamic DNS provider list. http://www.technopagan.org/dynamic/.

[5] eacceleration corporation. http://www.webcelerator.com/.

[6] Internet systems consortium. http://www.isc.org.

[7] Ircache home. http://www.ircache.net/.

[8] Lawrence berkeley national laboratory. http://ita.ee.lbl.gov/.

[9] Netscape, inc. http://www.netscape.com/.

[10] Standard performance evaluation co. http://www.specbench.org/.

[11] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May/June 2000.

[12] M. F. Arlitt and C. L. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631645, October 1997.

[13] G. Banga, F. Douglis, and M. Rabinovich. Optimistic deltas for www latency reduction. In *Proceedings of the USENIX Technical Conference*, 1997.

[14] G. Banga and P. Druschel. Measuring the capacity of a web server under realistic loads. *World Wide Web Journal*, 2(1-2):69–83, 1999.

[15] N. Bansal and M. Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. In *Proceedings of ACM Sigmetrics*, 2001.

[16] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns. *World Wide Web Journal*, 2(1):15–28, January 1999.

[17] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance '98/SIGMETRICS'98*, Madison, Wisconsin USA, July 1998.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[18] F. BASKETT, K. CHANDY, R. MUNTZ, AND F. PALACIOS. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2), Apr. 1975.

[19] A. BESTRAVOS. 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*, Baltimore, Maryland, 1995.

[20] R. BHAGWAN, S. SAVAGE, AND G. VOELKE. Understanding availability. In *Proceedings of IPTPS'03*, Berkeley, CA, USA, Feb. 2003.

[21] S. BLAKE, D. BLACK, M. CARLSON, E. DAVIS, Z. WANG, AND W. WEISS. An architecture for differentiated services. http://info.internet.isi.edu/in-notes/rfc/files/rfc2475.txt, December 1998.

[22] T. BRAY. Measuring the web. In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, 1996.

[23] L. BRESLAU, P. CAO, L. FAN, G. PHILLIPS, AND S. SHENKER. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM*, New York, NY, March 1999.

[24] S. BRIN AND L. PAGE. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998.

[25] A. BROIDO, E. NEMETH, AND K. CLAFFY. Spectroscopy of DNS update traffic. In *Proceedings of ACM SIGMETRICS'2003*, San Diego, CA, June 2003.

[26] N. BROWNLEE, K. CLAFFY, AND E. NEMETH. DNS Root/gTLD performance measurements. In *Proceedings of USENIX LISA'2001*, San Antonio, TX, December 2001.

[27] R. CACERES, F. DOUGLIS, A. FELDMANN, G. GLASS, AND M. RABINOVICH. Web proxy caching: the devil is in the details. In *Proceedings of the Workshop on Internet Server Performance*, Madison, Wisconsin, June 1998.

[28] M. CASTRO, P. DRUSCHEL, A. KERMARREC, AND A. ROWSTRON. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications*, volume 20, Oct. 2002.

[29] M. CASTRO, P. DRUSCHEL, A-M. KERMARREC, A. NANDI, A. ROWSTRON, AND A. SINGH. Splitstream: High-bandwidth multicast in a cooperative environment. In *Proceedings of the 19th ACM SOSP'03*, Lake Bolton, NY, USA, Oct. 2003.

[30] V. CATE. Alex - a global file system. In *Proceedings of USENIX File System Workshop'92*, Ann Arbor, MI, May 1992.

[31] A. CHANKDUNTHOD, P. DANZIG, C. NEERDAELS, M. SCHWARTZ, AND K. WORRELL. A hierarchical internet object cache. In *Proceedings of USENIX Annual Technical Conference'96*, San Diego, CA, January 1996.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[32] X. CHEN, H.N. WANG, S.S. REN, AND X. ZHANG. Strong cache consistency support for domain name system. In *Technical Report TR-04-12*, College of William and Mary, August 2004.

[33] X. CHEN AND X. ZHANG. Popularity-based ppm: An effective web prefetching technique for high accuracy and low storage. In *Proceedings of the 2002 International Conference on Parallel Processing*, Vancouver, Canada, August 2002.

[34] X. CHEN AND X. ZHANG. A popularity-based prediction model for web prefetching. *IEEE Computer*, March 2003.

[35] Y. CHEN, R.H. KATZ, AND J.D. KUBIATOWIC. Scan: A dynamic, scalable, and efficient content distribution network. In *Proceedings of the First International Conference on Pervasive Computing*, Zurich, Switzerland, Aug. 2002.

[36] Y. CHEN, L. QIU, W. CHEN, L. NGUYEN, AND R. H. KATZ. Clustering web content for efficient replication. In *Proceeding of the 10th IEEE International Conference on Network Protocols*, Paris, France, Nov. 2002.

[37] J. CHO AND H. GARCIA-MOLINA. The evolution of the web and implications for an incremental crawler. In *Proceedings of 26th International Conference on Very Large Databases*, Cairo, Egypt, Sep. 2000.

[38] I. CLARKE, O. SANDBERG, B. WILEY, AND T.W. HONG. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[39] J. G. CLEARY AND I. H. WITTEN. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[40] E. COHEN AND H. KAPLAN. Prefetching the means for document transfer: A new approach for reducing web latency. In *Proceedings of IEEE INFOCOM*, Tel Aviv, Israel, 2000.

[41] E. COHEN AND H. KAPLAN. Proactive caching of dns records: Addressing a performance bottleneck. In *Proceedings of The 2001 Symposium on Applications and the Internet*, San Diego, LA, January 2001.

[42] E. COHEN AND H. KAPLAN. Proactive caching of DNS records: Addressing a performance bottleneck. In *Proceedings of IEEE Symposium on Applications and the Internet'2001*, San Diego, CA, January 2001.

[43] E. COHEN, H. KAPLAN, AND U. ZWICK. Connection caching. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999.

[44] 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*, 1998.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[45] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a peer-to-peer lookup service. In *Proceedings of IPTPS'2002*, Cambridge, MA, March 2002.

[46] C. Cranor, E. Gansner, B. Krishnamurthy, and O. Spatscheck. Characterizing large DNS traces using graphs. In *Proceedings of ACM IMW'2001*, San Francisco, CA, November 2001.

[47] M. Crovella and P. Barford. Self-similarity in world wide web traffic: evidence and possible causes. In *Proceddings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer systems*, may 1996.

[48] M. Crovella and P. Barford. The network effects of prefetching. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, April 1998.

[49] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[50] F. M. Cuenca-Acuna, R. P. Martin, and T.D. Nguyen. Autonomous replication for high availability in unstructured p2p systems. In *Proceedings of IEEE SRDS'03*, Florence, Italy, Oct. 2003.

[51] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the 18th ACM SOSP'01*, Banff, Alberta, Canada, Oct. 2001.

[52] P. Danzig, K. Obraczka, and A. Kumar. An analysis of wide-area name server traffic: A study of the internet domain name system. In *Proceedings of ACM SIGCOMM'92*, Baltimore, MD, August 1992.

[53] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *Proceedings of IEEE ICDCS'03*, Providence, RI, USA, May 2003.

[54] B. D. Davison. Predicting web actions from html content. In *Proceedings of the The Thirteenth ACM Conference on Hypertext and Hypermedia*, College Park, MD, June 2002.

[55] B. D. Davison and Vincenzo Liberatore. Pushing politely: Improving web responsiveness one packet at a time. *Performance Evaluation Review*, 28(2):43–49, September 2000.

[56] D. Duchamp. Prefetching hyperlinks. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, October 1999.

[57] V. Duvvuri, P. Shenoy, and R. Tewari. Adaptive leases: A strong consistency mechanism for the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 15(5), September/October 2003.

[58] D. Eastlake. Domain name system security extensions. In *RFC 2535*, March 1999.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[59] R. BRADEN ED., L. ZHANG, S. BERSON, S. HERZOG, AND S. JAMIN. Resource reservation protocol (rsvp) - version 1 function specification. http://info.internet.isi.edu/in-notes/rfc/files/rfc2205.txt, September 1997.

[60] JON EISENBERG AND CRAIG PARTRIDGE. The internet under crisis conditions: Learning from september 11. *ACM Computer Communication Review*, 33(2), April 2003.

[61] 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*, May 1999.

[62] B. GEDIK AND L. LIU. Reliable peer-to-peer information monitoring through replication. In *Proceedings of IEEE SRDS'03*, Florence, Italy, Oct. 2003.

[63] C. G. GRAY AND D. R. CHERITON. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of ACM SOSP'89*, Litchfield Park, AZ, December 1989.

[64] WCOL GROUP. Www collector: the prefetching proxy server for www. http://shika.aist-nara.ac.jp/products/wcol/, 1997.

[65] J.L. HERLOCKER AND J.A. KONSTAN. Content-independent task-focused recommendation. *IEEE Internet Computing*, pages 40–47, November/December 2001.

[66] B. A. HUBERMAN, P. L. T. PIROLLI, J. E. PITKOW, AND R. M. LUKOSE. Strong regularities in world wide web surfing. *Science*, 280:95–97, April 1998.

[67] A. K. IYENGAR, E. A. MACNAIR, M. S. SQUILLANTE, AND L. ZHANG. A general methodology for characterizing access patterns and analyzing web server performance. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Montreal, Canada, July 1998.

[68] S. IYER, A. ROWSTRON, AND P. DRUSCHEL. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of ACM PODC'02*, Monterey, California, USA, July 2002.

[69] Z. JIANG AND L. KLEINROCK. An adaptive network prefetch scheme. *IEEE Journal on Selected Areas of Communication*, 17(4):358–368, 1998.

[70] D. JOSEPH AND D. GRUNWALD. Prefetching using markov predictors. *IEEE Transactions on Computers*, 48(2), Februray 1999.

[71] J. JUNG, D. LEE, AND K. CHON. Proactive web caching with cumulative prefetching for large multimedia data. In *Proceeding of 9th International World Wide Web Conference*, 2000.

[72] J. JUNG, E. SIT, H. BALAKRISHNAN, AND R. MORRIS. DNS performance and the effectiveness of caching. In *Proceedings of ACM IMW'2002*, San Francisco, CA, October 2002.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[73] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of ACM STOC'97*, El Paso, TX, USA, May 1997.

[74] J. I. Khan and Q. Tao. Partial prefetch for faster surfing in composite hypermedia. In *USENIX Symposium on Internet Technology and Systems*, San Francisco, CA, March 2001.

[75] R. P. Klemm. Webcompanion: A friendly client-side web prefetching agent. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):577594, July/August 1999.

[76] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin. A non-interfering deployable web prefetching system. In *USENIX Symposium on Internet Technology and Systems*, Seattle, WA, March 2003.

[77] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploiting the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, April 1997.

[78] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, and D. Geels. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS-IX*, Cambridge, MA, USA, Nov. 2000.

[79] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham. Consistency maintenance in peer-to-peer file sharing networks. In *Proceedings of IEEE WIAPP'03*, San Jose, CA, USA, June 2003.

[80] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. In *Proceedings of VLDB 2002*, Hong Kong, China, Aug. 2002.

[81] R. Liston, S. Srinivasan, and E. Zegura. Diversity in DNS performance measures. In *Proceedings ACM IMW'2002*, Marseille, France, November 2002.

[82] C. Liu and P. Cao. Maintaining strong cache consistency in the World-Wide Web. *IEEE Transactions on Computers*, 47(4):455–457, April 1998.

[83] E. P. Markatos and C. E. Chronaki. A top-10 approach to prefetching on the web. *Technical Report, No. 173*, August 1996.

[84] D. A. Menasc and V. A. F. Almeida. Capacity planning for web services: metrics, models, and methods. *Prentice Hall, NJ*, 2002.

[85] M. Mikhailov and C. Wills. Evaluating a new approach to strong web cache consistency with snapshots of collected content. In *Proceedings of WWW'2003*, Budapest, Hungary, May 2003.

[86] P. Mockapetris. Domain names-concepts and facilities. In *RFC1034*, November 1987.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[87] P. MOCKAPETRIS. Domain names-implementation and specification. In *RFC 1035*, November 1987.

[88] D. MOSBERGER AND T. JIN. httperf - a tool for measuring web server performance. *Performance Evaluation Review*, 26(3):31–37, Dec. 1998.

[89] A. NTOULAS, J. CHO, AND C. OLSTON. What's new on the web? the evolution of the web from a search engine perspective. In *Proceedings of the World Wide Web Conference*, New York, NY, USA, May 2004.

[90] D. OLSHEFSKI, J. NIEH, AND D. AGRAWAL. Inferring client response time at the web server. In *Proceedings of SIGMETRICS*, 2002.

[91] V. N. PADMANABHAN AND J. C. MOGUL. Using predictive prefetching to improve world wide web latency. *Computer Communication Review*, 1996.

[92] J. PANG, A. AKELLA, A. SHAIKH, B. KRISHNAMURTHY, AND S. SESHAN. On the responsiveness of DNS-based network control. In *Proceedings of ACM IMC'2004*, Taormina, Sicily, Italy, October 2004.

[93] J. PANG, J. HENDRICKS, A. AKELLA, R. DE PRISCO, B. MAGGS, AND S. SESHAN. Availability, usage and deployment characterisitics of the domain name system. In *Proceedings of ACM IMC'2004*, Taormina, Sicily, Italy, October 2004.

[94] V. PAPPAS, Z. XU, S. LU, A. TERZES, D. MASSEY, AND L. ZHANG. Impact of configuration errors on DNS robustness. In *Proceedings of ACM SIGCOMM'2004*, Portland, OR, August 2004.

[95] K. PARK, V. S. PAI, L. PETERSON, AND Z. WANG. CoDNS: Improving DNS performance and reliability via cooperative lookups. In *Proceedings of USENIX OSDI'2004*, San Francisco, CA, December 2004.

[96] D. A. PATTERSON. Latency lags bandwidth. *Communications of the ACM*, 47:71–75, Oct. 2004.

[97] V. PAXSON AND S. FLOYD. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[98] J. PITKOW AND P. PIROLLI. Mining longest repeating subsequences to predict world wide web surfing. In *Proceedings of the 1999 USENIX Technical Conference*, April 1999.

[99] V. RAMASUBRAMANIAN AND E.G. SIRER. Beehive: Exploiting power law query distributions for o(1) lookup performance in peer to peer overlays. In *Proceedings of USENIX NSDI'04*, San Francisco, CA, USA, Mar. 2004.

[100] V. RAMASUBRAMANIAN AND E.G. SIRER. The design and implementation of a next generation name service for the internet. In *Proceedings of ACM SIGCOMM'2004*, Portland, Oregon, USA, August 2004.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[101] V. Ramasubramanian and E.G. Sirer. The design and implementation of a next generation name service for the internet. In *Proceedings of ACM SIGCOMM'04*, Portland, OR, USA, Aug. 2004.

[102] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, USA, Aug. 2001.

[103] Y. Rekhter, S. Thomson, J. Bound, and P. Vixie. Dynamic updates in the domain name system. In *RFC2136*, April 1997.

[104] M. Roussopoulos and M. Baker. Cup: Controlled update propagation in peer-to-peer networks. In *Proceedings of 2003 USENIX Annual Technical Conference*, San Antonio, Texas, USA, June 2003.

[105] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware'01*, Heidelberg, Germany, Nov. 2001.

[106] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale persistent peer-to-peer storage utility. In *Proceedings of ACM SOSP'01*, Banff, Alberta, Canada, Oct. 2001.

[107] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the pangaea wide-area file system. In *Proceedings of USENIX OSDI'02*, Boston, Massachusetts, USA, Dec. 2002.

[108] R. Sarukkai. Link prediction and path analysis using markov chains. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, Netherlands, May 2000.

[109] S. Schechter, M. Krishnan, and M. D. Smith. Using path profiles to predict http requests. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998.

[110] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proceedings of IEEE INFOCOM'2001*, Anchorage, AK, April 2001.

[111] S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements. http://info.internet.isi.edu/in-notes/ref/files/rfc2215.txt, September 1997.

[112] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proceedings of ACM MOBICOM'2000*, Boston, MA, August 2000.

[113] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, USA, Aug. 2001.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[114] G. Trent and M. Sake. Webstone: the first generation in http server benchmarking. In *Silicon Graphics White Paper*, Febuary 1995.

[115] A. Venkataramani, R. Kokku, and M. Dahlin. System support for background replication. In *Proceedings of Fifth Operating Systems Design and Implementation conference*, Boston, MA, December 2002.

[116] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the web from dns. In *Proceedings of USENIX NSDI'2004*, San Francisco, CA, USA, March 2004.

[117] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the web from dns. In *Proceedings of USENIX NSDI'04*, San Francisco, CA, USA, Mar. 2004.

[118] Z. Wang and J. Crowcroft. Prefetching in world-wide web. In *Proceedings of IEEE Globecom*, London, UK, December 1996.

[119] B. Wellington. Secure domain name system dynamic update. In *RFC3007*, November 2000.

[120] D. Wessels, M. Fomenkov, N. Brownlee, and K. Claffy. Measurement and laboratory simulations of the upper DNS hierarchy. In *Proceedings of PAM'2004*, Antibes Juan-les-Pins, France, April 2004.

[121] C. Wills, M. Mikhailov, and H. Shang. Inferring relative popularity of internet applications by actively querying DNS caches. In *Proceedings of ACM IMC'03*, Miami, FL, October 2003.

[122] C. Wills and H. Shang. The contribution of DNS lookup costs to web object retrieval. In *Technical Report TR-00-12*, Worcester Polytechnic Institute, July 2002.

[123] L. Xiao and X. Zhang. Exploiting neglected data locality in browsers (poster). In *Proceedings of 10th International World Wide Web Conference*, 2001.

[124] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Using leases to support server-driven consistency in large-scale systems. In *Proceedings of IEEE ICDCS'98*, Amsterdam, Netherlands, May 1998.

[125] J. Yin, M. Dahlin, L. Alvisi, C. Lin, and A. Iyengar. Engineering server driven consistency for large scale dynamic web services. In *Proceedings of WWW'2001*, Hong Kong, China, May 2001.

[126] H. Yu and A. Vahdat. Consistent and automatic service regeneration. In *Proceedings of USENIX NSDI'04*, San Francisco, CA, USA, Mar. 2004.

[127] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. In *IEEE Journal on Selected Areas in Communications*, volume 22, Jan. 2004.

[128] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of ACM NOSSDAV'01*, Port Jefferson, NY, USA, June 2001.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

VITA

Xin Chen

Xin Chen received the BS and MS degrees in computer science from Xi'an Jiaotong University and University of Science and Technology of China, in 1996 and 1999, respectively. He is a PhD candidate of computer science at the College of William and Mary. His research interests are distributed systems, networking and Internet computing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.