

## TABLE OF CONTENTS

|   |      |
|---|------|
| ACKNOWLEDGEMENT.....                              | viii |
| CHAPTER   |      |
| 1 Introduction.....                               | 1    |
| 1.1 Related Works.....                            | 2    |
| 1.2 About this thesis.....                        | 5    |
| 2 Impact Factors for Prefetching Performance..... | 8    |
| 2.1 Reading Behaviors.....                        | 8    |
| 2.2 Web Organizations.....                        | 9    |
| 3 The Architecture of RHDOS.....                  | 17   |
| 3.1 Hypertext Transfer Protocol (HTTP).....       | 17   |
| 3.2 HTTP Proxy.....                               | 18   |
| 3.3 RHDOS Transaction.....                        | 23   |
| 3.4 Recording Time for Implement Event.....       | 26   |
| 4 Simulation Experiment.....                      | 30   |
| 4.1 Overview.....                                 | 30   |
| 4.2 RHDOS implemen.....                           | 31   |
| 4.3 Performance Results Analysis.....             | 32   |
| 4.3.1 Chain.....                                  | 32   |
| 4.3.2 Tree.....                                   | 37   |
| 4.3.3 Fully Connected Graph.....                  | 50   |
| 4.3.4 Tree with Core Graph.....                   | 58   |
| 5 Conclusions and Future Works.....               | 64   |
| 5.1 Conclusions.....                              | 64   |
| 5.2 Future Works.....                             | 65   |
| References.....                                   | 67   |

Appendix A Lists of Some Examples for Web pages Organization..... 70

## **CHAPTER 1**

### **Introduction**

Internet is a global distributed, dynamic information repository that contains vast amount of digitized information. Most Web pages not only contain a simple parent HTML file with few embedded images, but also contains embedded entities such as banners, Java applets, flash presentations, etc. with varying rendering constraints.

Although the core network speed is doubled every 9-12 months, the network traffic is growing even faster. Because the growth in the last mile speed is slower and the multimedia traffic such as audio, video, and images are also increased, the Web response delay is still increasing.

There are two principal techniques for speeding up access, which include Web caching and Web prefetching.

Over last 5 years the Web caching techniques have matured and been deployed.

This thesis will focus on how Web prefetching technique can be used effectively to improve responsiveness of web systems. We define Web prefetching as follows:

While a user is looking at the current Web page, we try to predict what are the next pages the user will most likely view and prefetch them.

However, simple prefetching seems to be limited because of excessive transfer of unused bytes. In this thesis, we will compare some important factors, such as reading habit, reading time, and prefetching sequence to see how they affect the performance of

prefetching.

## **1.1 Related Works**

A number of recent researches have anticipated that prefetching can significantly enhance Web response just like it has accelerated hardware system.

In one of the pioneering studies, Kroeger et al. [KrLM97] used traces of Web proxy activity to find out that the external latency between the proxy cache and the server accounts for 77% of the total latency. Local proxy caching could reduce latency by at most 26%. Prefetching could reduce latency by at most 57%. A combined caching and prefetching proxy could provide at most a 60% latency reduction. Furthermore, they found that how far in advance a prefetching algorithm was able to prefetch an object was a significant factor in its ability to reduce latency. They observed that prefetching lead time is an important factor in the performance of prefetching and prefetching can offer more than twice the improvement of caching but is still limited in its ability to reduce latency.

Prefetching technique was also applied in improving the performance of the World Wide Web over wireless links [FIMD97]. Fleming developed a system that consists of a proxy server and a modified client and used a new protocol called MHSP. The proxy prefetches documents to the client, which improves performance over high bandwidth links, enhances scheme substantially, reduces error rates by 40%, saves network bandwidth by 13.18%, and increases byte hit rates by 8.1% for document availability

when the connection is broken due to wireless effects. This system reduces document load time by 32% to 37% when compared to HTTP.

Jacobson and Cao [JaCa98] proposed a prefetching method based on partial context matching technique between low bandwidth clients and proxies. This work showed that prefetching could reduce latency by less than 10% (predicting 12% of the requests, and increasing traffic by 18%). A significant part of this reduction is attributed to the caching effect of the prefetching buffer. Palpanas and Mendelzon [PaMe99] demonstrated that a k-order Markov predictor scheme similar to those used in image compression can reduce response time by a factor of up to 2. Both these methods used variants of partial matching of context (past sequence of accessed references) for prediction of future Web reference. These works suggested prefetching in order of highest likelihood of access.

Crovella and Bradford [CrBa98] studied another advantage of prefetching. A trace driven simulation indicates that straightforward approaches to prefetching increase the bursting of traffic. Instead, the authors propose a transport rate control mechanism. The simulation denotes that rate-controlled prefetching significantly improves network performance compared not only with the straightforward approach, but also with the non-prefetching case, while delivering the requested documents on time.

Pitkow and Pirolli [PiPi99] investigated various methods that have evolved to predict surfer's path from log traces such as session time, frequency of clicks, Levenshtein Distance analyses and compared the accuracy of various construction methods. This Markov model based study noted that although information is gained by

studying longer paths, conditional probability estimate, given the surf path, is more stable over time for shorter paths and can be estimated reliably with less data.

In related work, Duchamp [Duch99] discussed a method for clients and servers to exchange information. Its features included: how information on access patterns is shared by the server over clients; occurs in near-real time; is configurable by client; many previously uncachable pages can be prefetched; both client and server can cap operations to limit impact on overhead and bandwidth; and it operates as an HTTP extension. The overall results were very positive: client latency improved greatly (slightly over 50%), and less of the cache was wasted (about 60% of prefetched pages were eventually used).

Cohen and Kaplan [CoKa00] cited problems from bandwidth waste in prefetching, and as opposed to document prefetching, they suggested pre-staging only the communication session- such as pre-resolving DNS, pre-establishing of TCP connection and pre-warming by sending dummy HTTP HEAD request. RealPlayer (release 8) already pre-stages streaming connections linked from a page by pre-extracting and readying individual coded information associated with each.

However, some experts suspect the advantage of using indiscriminate prefetching technique [Davi01, Khan00, Khan99, KaPJ99].

Kaashoek [KaPJ99] traced Web server to find that, on average, 0.5 objects are prefetched for each object explicitly fetched by the user. Among these prefetched objects, only about 2% are actually used in the future. The others just waste bandwidth and unnecessarily load servers.

Khan [Khan99, Khan00] demonstrated that instead of simply ranking candidate hyperlinks in order of transition probabilities – a ranking order that also considers the loading time can yield much better performance with respect to larger prediction error.

Brian D. Davison [Davi01] argues that the current support for prefetching in HTTP/1.1 is insufficient using HTTP GET. Existing prefetching implementations can cause problems with undesirable side effects and server abuse, and the potential for these problems may thwart additional prefetching development and deployment. They make some suggestions for HTTP that would allow for safe prefetching, reduced server abuse, and differentiated Web server quality of service.

To reduce unnecessary prefetching, Khan and Tao [KhTa01] suggested a “partial prefetching” mechanism for composite multimedia documents. Each composite multimedia page and its components are optimally divided into two parts, the lead segments and the stream segments. The system always loads two parallel streams. In operation only the lead segment of candidate is prefetched, the stream segment of the current document is fetched as necessary. Simulation results were presented based on statistical models that project the scheme’s performance under varying conditions and reported the maximum improvement in system responsiveness is about 3.6 times.

## **1.2 About this thesis**

Prefetching has been a major focus of recent research in World Wide Web, since the prefetching prediction model can reduce the access lag. However, most of the current prefetching models also result in excess transfer of unused data. It seems more

innovations are required before prefetching technology can reach a mature point. A study of the recent works shows that most of the suggested prefetching techniques took a statistical approach to the problem. Almost all of the suggested works concentrated on a concept of “access frequency” as the principle guideline for the prefetching activities. Many researchers come up with varying techniques for estimation and/or its prediction.

Interestingly these studies do not take into consideration two intriguing aspects. From the very top level a web system is a conduit of communication between two principle parties – the content developer and the content reader. The intermediate components – the server, the browser and the proxy work as a facilitator in this communication. It seems therefore almost natural that the prefetching performance should be strongly dependent on the intent and behavior of the two principles. Roughly speaking this points out that on one hand the nature and organization of the content, and on the other hand, the reading and interaction style of the reader should have important impact on the prefetching performance. The topic of readers’ behavior has been absent from previous work. Also, the concept of “access frequency” does not sufficiently capture the various patterns in the content organization. The intent of this thesis is to shed some light in this void. The thesis is an attempt to study how content organization and reading habit can affect the prefetching performance of a web system.

The next chapter presents the impact factors for prefetching performance. Chapter 3 focuses on the working scheme of HTTP proxy and the architecture of RHDOS. Chapter 4 is the main body of this thesis. The performance of prefetching is evaluated in details

and followed by the analysis of performance results. The last chapter provides a summary of this thesis including some thoughts about the future work.

## CHAPTER 2

### Impact Factors for Prefetching Performance

#### 2.1 Reading Behaviors

The reason we need to analyze reading habits is to accurately predict the Web contents readers want to move through. However, it is quite complex to model the readers' behavior. Different reader has different text reading speed. It also depends on the content type. Also, various readers may have different psychological pattern guiding their browsing habit. For example, in the case of reading an online e-book, different readers view them in different reading sequence. After finishing reading the instruction for chapter 1, some readers may continue reading section 1 of chapter 1, and other may skip to the instruction for chapter 2. The question is which option should be prefetched? If we prefetch both of them, which one should come first? Different answers will certainly result in different performance results for prefetching. The detail modeling of the user behavior is quite complex. However, the goal of this thesis is to capture the essence, and limit the study on few but core parameters. We consider two parameters as below:

- Reading sequence
- Reading time as elements of user interaction habit

Reading time can be defined as the time a reader spends on a certain page. It is the viewing duration or interaction time between the time users receive the first request and

send out the second request. For the purpose of analyzing the prefetching performance, we call it interaction interval. Usually, the more reading time readers spend on each Web page, the more prefetched Web pages they could acquire.

This thesis demonstrated in detail that the difference in these two could have significant impact on prefetching performance.

## **2.2 Web Organizations**

Today Web pages are becoming more and more sophisticated. Web designers tend to pay a lot of efforts to develop an appealing user interface. They often do not know that through the implementation of prefetching technology, they can change the organization of their Web pages to reduce user access time. In fact, the organization of Web structure can have tremendous impact on prefetching performance. It decides how prefetching can be implemented and how much data have to be involved.

The modeling of content organization is also not trivial. Current web contents come in various complex organizations. However, an analysis of recent web documents seems to suggest that although there is no concrete discipline, but few patterns do tend to emerge. From Fig. 2.1 to Fig. 2.5, we display some Web documents. Each of them is organized in a unique way. In our modeling process, we therefore selected few patterns, which tend to emerge more frequently. These patterns do not appear in ideal format but in the generalized graph a significant sub-graph tends to contain two major patterns tree, and chain. However, more recent pages also show fully connected sub graph component. Fig. 2.1 and Fig. 2.2 are an example for Web-based quizzes. Fig. 2.3 is a photo album.

On each page we click Next to move on. The surfer seems to be moving through a form of sequential chains. These Web pages have a chain structure.

The Web page in Fig.2.4 groups WWW knowledge into different categories. It uses a tree structure, and each category is a node in tree. Fig. 2.5 shows an example for online encyclopedia. Readers can easily move back and forth through any Web pages, no matter what the current page is. Each Web page is connected with each other. We consider this type of organization as a fully connected graph. Therefore, we provide the study on these three document organizations. Some samples are shown in table 2.1 and Appendix A.

However, in a higher level, the patterns seem to be a combination of several of these core patterns. One of the more prevalent forms of complex pattern is combination of fully connected section and tree. This type of Web page usually relies on a frame set. One component of the frameset is a menu that remains visible as links to other pages. Therefore, we include a forth study on a complex pattern called a “tree with a fully connected core”. In this experiment, according to the organization approach, we therefore study the following impact of four organizations:

- Chain
- Tree
- Fully connected graph
- Tree with core graph

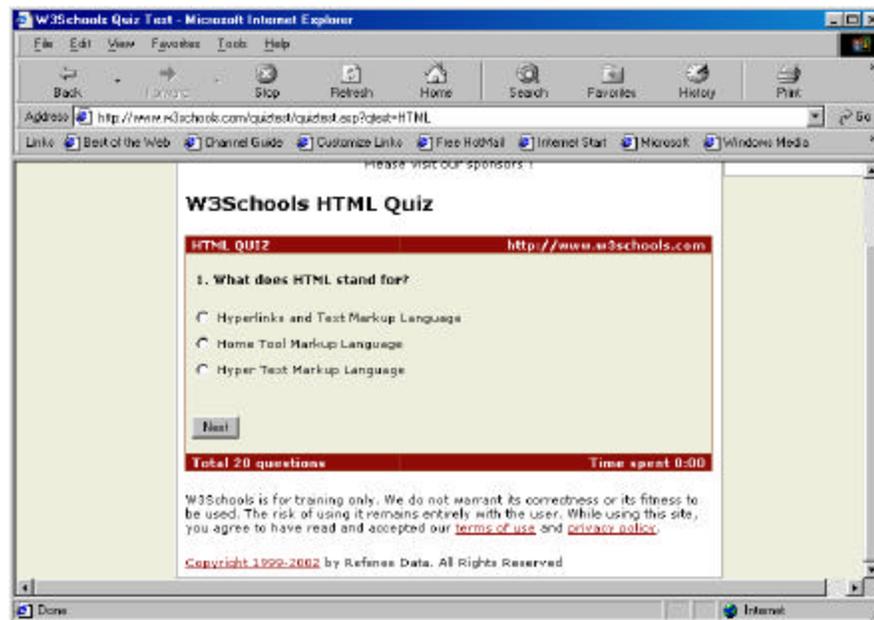


Fig. 2.1 An Example of Chain in Web-based Quiz (1)

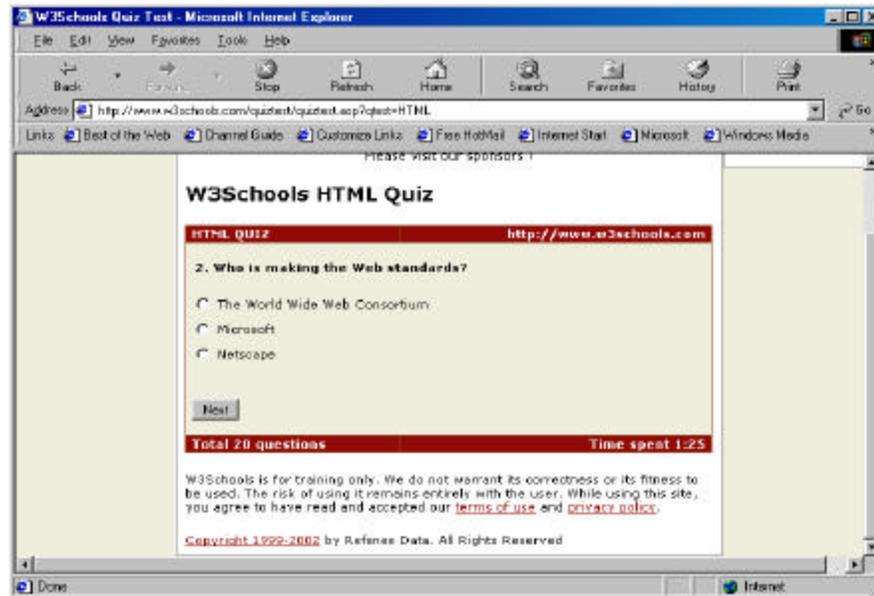


Fig. 2.2 An Example of Chain in Web-based Quiz (2)

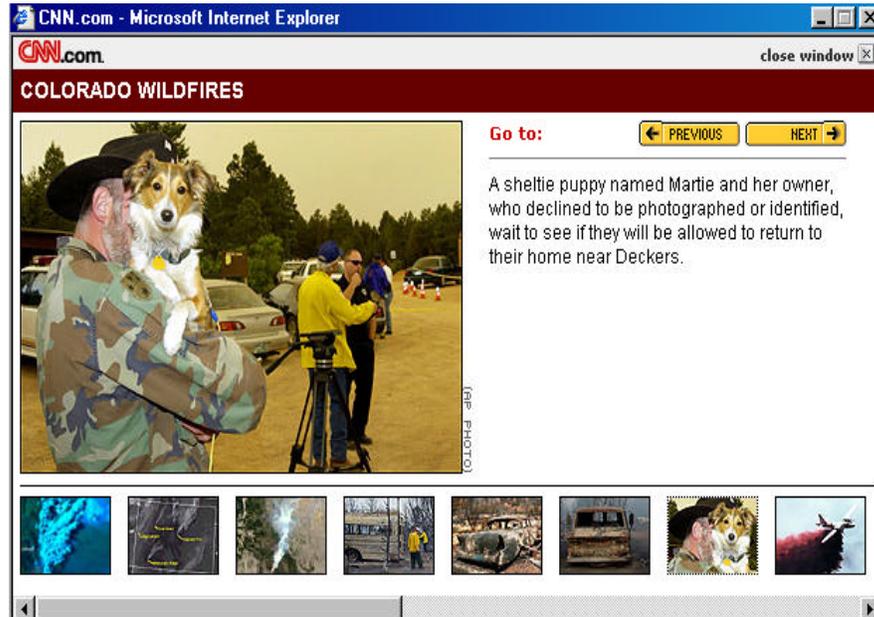


Fig. 2.3 An Example of Chain in Photo Album

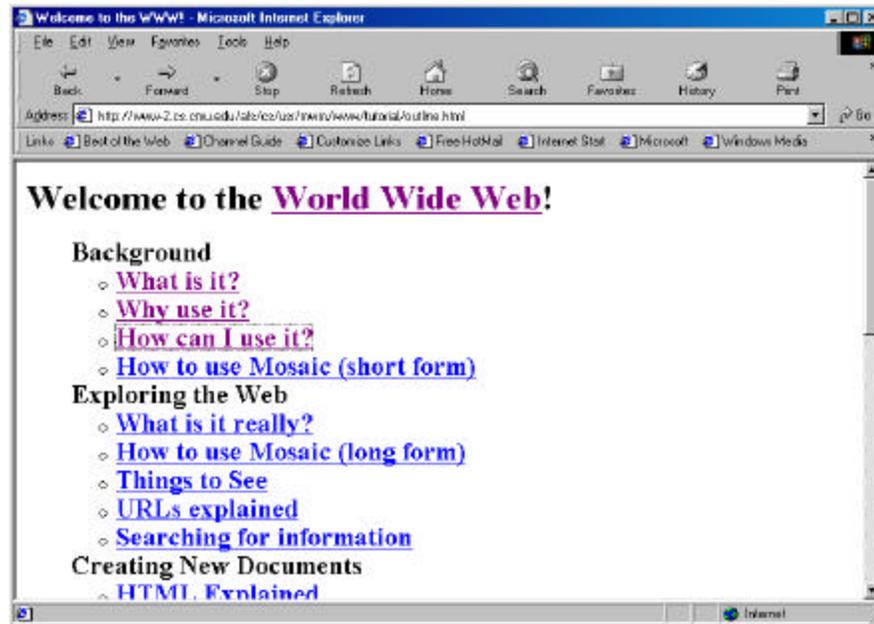


Fig. 2.4 An Example of Tree in the Categories of Tutorial

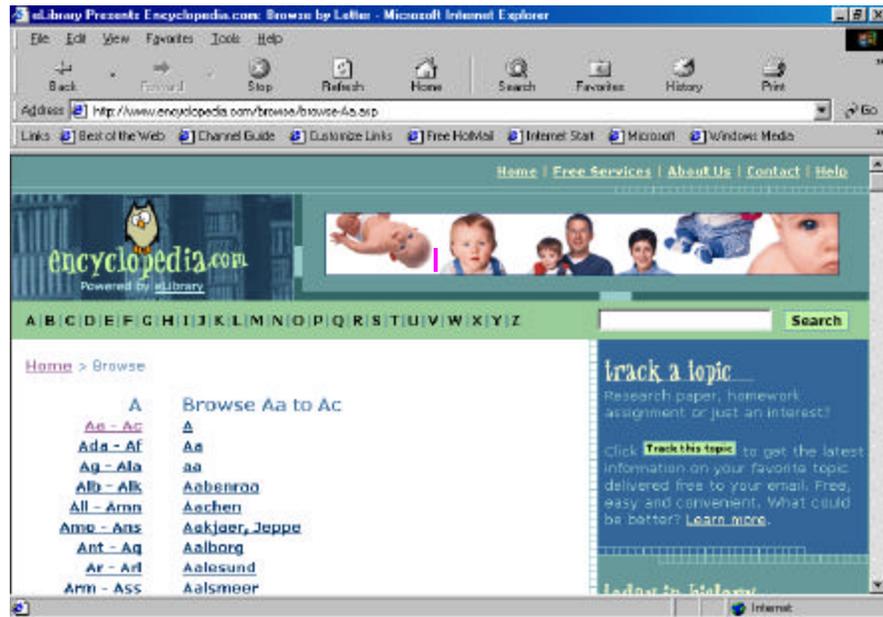


Fig. 2.5 An Example of Fully Connected Graph in the Encyclopedia

| Contents      | Type            | URL   |
|---------------|-----------------|---|
| Quiz          | Chain           | <a href="http://www.w3schools.com/quiztest/quiztest.asp?qtest=HTML">http://www.w3schools.com/quiztest/quiztest.asp?qtest=HTML</a>                 |
| Photo Album   | Chain           | <a href="http://www.cnn.com/2002/US/07/01/western.wildfires/index.html">http://www.cnn.com/2002/US/07/01/western.wildfires/index.html</a>         |
| Registration  | Chain           | <a href="http://www.ingenta.com/isis/register/RegisterPersonalUser/ingenta">http://www.ingenta.com/isis/register/RegisterPersonalUser/ingenta</a> |
| Categories    | Tree            | <a href="http://www.cs.kent.edu">http://www.cs.kent.edu</a>   |
| Tutorial      | Tree            | <a href="http://www-2.cs.cmu.edu/afs/cs/usr/mwm/www/tutorial/outline.html">http://www-2.cs.cmu.edu/afs/cs/usr/mwm/www/tutorial/outline.html</a>   |
| Homepage      | Fully Connected | <a href="http://www.kent.edu/academics/">http://www.kent.edu/academics/</a>   |
| Encyclopedia  | Fully Connected | <a href="http://www.encyclopedia.com/browse/browse-Aa.asp">http://www.encyclopedia.com/browse/browse-Aa.asp</a>                                   |
| E-book        | Fully Connected | <a href="http://safari.informit.com/?XmlId=0-13-084466-7">http://safari.informit.com/?XmlId=0-13-084466-7</a>                                     |
| Categories    | Tree with Core  | <a href="http://www.cnn.com/SHOWBIZ/">http://www.cnn.com/SHOWBIZ/</a>   |
| Search Engine | Tree with Core  | <a href="http://www.google.com">http://www.google.com</a>   |

Table 2.1 Some Examples of Web pages Organization

## **CHAPTER 3**

### **The Architecture of RHDOS**

The objective of any prefetching system is to reduce user waiting time, and the potential cost factor is the amount of data is fetched which are never used. Therefore in this thesis we study the impact on the two performance measures:

- Response time
- The amount of data transfer

In order to observe and analyze the different performance results of prefetching, our own proxy system is called Reading Habit and Document Organization Sensitive Proxy or RHDOS. It is a HTTP proxy system designed to reduce latency while allowing the client to take advantage of available bandwidth.

#### **3.1 Hypertext Transfer Protocol (HTTP)**

HTTP [FMFM99] stands for Hypertext Transfer Protocol. It provides the foundation for the Web. HTTP has been used by the Web global information initiative since 1990. Initial version is HTTP /1.0. Its major drawback is it does not sufficiently take into consideration the effects of hierarchical proxies, caching, the demand for persistent connections, and virtual hosts.

Like other protocols, HTTP is constantly evolving. As of early 1997, the Web is moving from HTTP 1.0 to HTTP 1.1. It's more efficient overall, since it has addressed

new demands and overcome shortcomings of HTTP 1.0. In this thesis we refer to HTTP 1.1.

HTTP takes place through TCP/IP sockets. Like most network protocols, HTTP uses the client-server model: an HTTP client, in most cases a Browser, opens a connection and sends a request message to an HTTP server. HTTP defines the rules to phrase the requests. The server returns a response message, usually containing the resource that was requested. The rules of the response are also defined by HTTP. Therefore the HTTP protocol is a request/response protocol. Although the default port for HTTP servers to listen on is 80, they can use any port. After delivering the response, the server closes the connection. Fig. 3.1 shows a normal HTTP transaction.

HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by SMTP, NNTP, FTP, Gopher, and WAIS protocols. In this way, HTTP allows basic hypermedia access to resources available from diverse applications.

### **3.2 HTTP Proxy**

An HTTP proxy is a program that acts as an intermediary between a client and a server. After receiving requests from clients, it first attempts to find data locally, and if it's not there, fetches it from the remote server where the data resides permanently. The responses pass back through it in the same way. Thus, a proxy combines functions of both a client and a server.

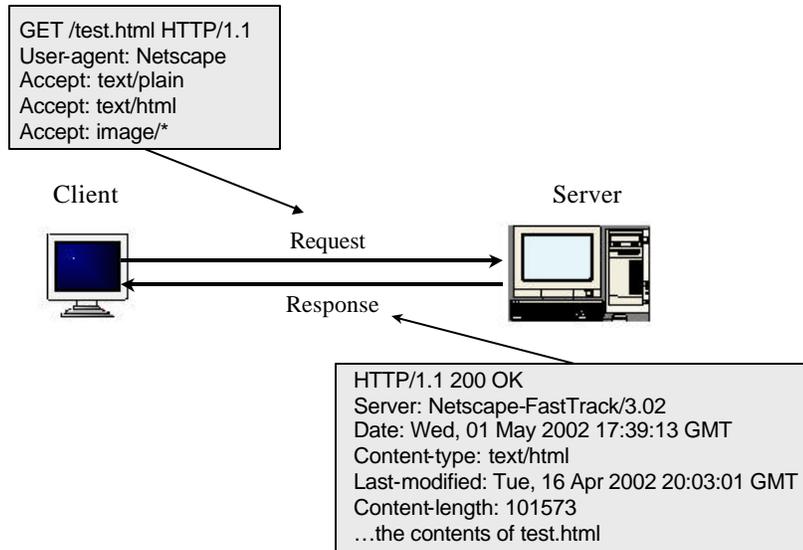


Fig. 3.1 The HTTP transaction between a client and a server

An HTTP proxy is an application-layer network service for caching Web objects. Unlike browser caches, a typical proxy can accept connections from multiple clients simultaneously and can connect to any sever. It is usually operated in the same way as other network services (e-mail, Web servers, DNS).

Proxies are commonly used in firewalls, LAN-wide caches, and other situations. This thesis will focus on the proxy, which involves caching. Fig. 3.2 shows how a caching proxy works. When a client uses a proxy, it typically sends a request to the proxy. The proxy connects to the HTTP server and the requested document is retrieved from the HTTP server and stored locally in the caching proxy for future use.

In Fig. 3.3, if an up-to-date version of the requested document is found in the cache, the caching proxy may be able to return it directly. No connection to the HTTP server is necessary.

A big problem with reusing copies of documents is keeping them up to date. If and when the original document is changed, the cached copy becomes inconsistent with the original and should not be used.

HTTP/1.1 uses the Age response-header to help convey age information between caches. The Age header value is the sender's estimate of the amount of time since the response was generated at the origin server.

In the case of a cached response that has been revalidated with the origin server, the Age value is based on the time of revalidation, not of the original response.

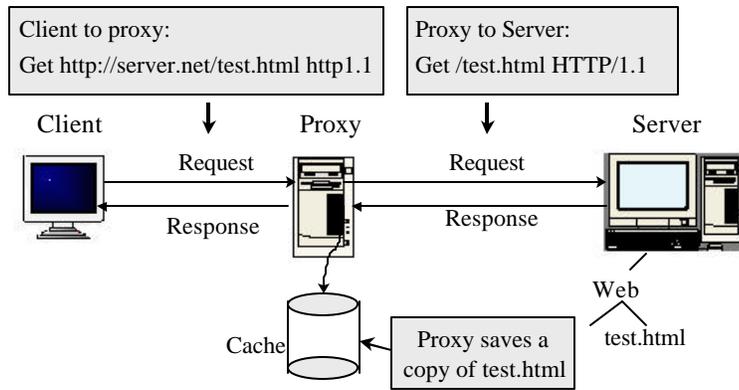


Fig.3.2 A Caching Proxy Transaction

Client to proxy:  
Get http://server.net/test.html http1.1

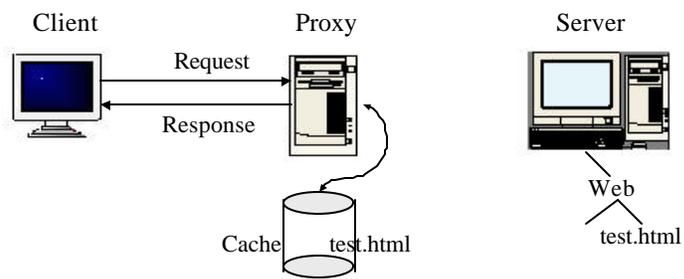


Fig. 3.3 Cache Hit On the Proxy

### **3.3 RHDOS Transaction**

The architecture for RHDOS is shown in Fig. 3.4. It is built upon an HTTP thread, which communicates with clients and servers; a cache thread, which stores and parses files; a prefetch thread, which retrieves Web documents in advance. RHDOS system is programmed in Java due to its powerful feature of multiple threads.

RHDOS includes five function modules: Proxy Caching Manager, New Item Request Manager, Document Analyzer, Priority Evaluator, and Loader. They are introduced individually in the following text.

#### **1. Proxy Caching Manager**

Proxy Caching Manager is the core and controller of RHDOS, which coordinates the other function modules. After initializing the proxy RHDOS system, a socket to listen to requests from client was created. A client initiates a connection to TCP via port 8080 on RHDOS. The port number could eventually be changed to any number in the system services range. Whenever the proxy RHDOS accepts a connection request, it starts a thread to handle the connection. Meanwhile, if a directory called cache is not available, RHDOS creates it and assigns its size. It also establishes a hash table for the cached files and hyperlinks message. Cached files are displayed with name, size, type, and modified time. Hyperlink message contains URL, the value of its frequency and estimated loading time.

After receiving a request from the client, Proxy Caching Manager parses the

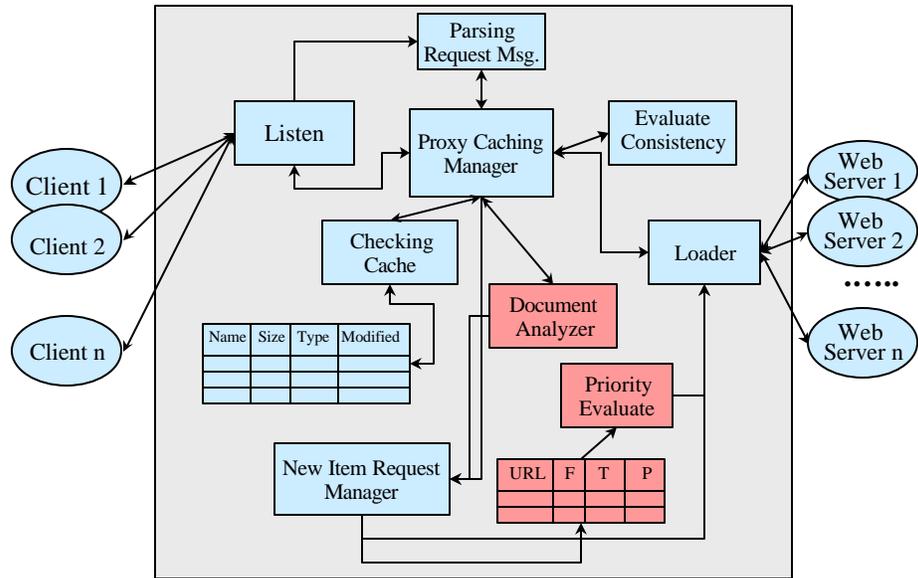


Fig. 3.4 Architecture of the RHDOS

request message and checks whether the request has been cached. If the Web document is already cached, it immediately hits cache and directly sends the document back to the client. Otherwise it tells New Item Request Manager to implement fetching task.

Proxy Caching Manager is also responsible for deleting the most outdated cached file until the proxy has enough space to cache newly received Web pages.

## 2. New Item Request Manager

If a cached file does not exist, New Item Request Manager needs to rebuild a request to send to the Web server. If the file does not exist, a “Not Found” message will be sent to the client. It also accepts some new HTTP requests, which come from the parsing result of Document Analyzer to implement prefetching tasks. The parsed hyperlinks message including URLs, the value of frequency and estimated loading time are stored in a hash table.

## 3. Document Analyzer

While Proxy Caching Manager sends a response back to the client, Document Analyzer parses and extracts all hyperlinks from the requested document. The parsed information is sent to New Item Request Manager to implement prefetching tasks.

## 4. Priority Evaluator

Priority Evaluator is responsible for calculating all URL priorities ( $P_i$ ) according to formula 3.1.  $F_i$  means frequency and  $T_i$  the estimated loading time. They come from

Document Analyzer and have been stored in the hash table. The value of  $P_i$  will be provided to Loader as prefetching sequence.

$$P_i = F_i / T_i \quad \dots\dots\dots 3.1$$

## 5. Loader

The job of Loader function is to fetch Web pages from Web server. Web pages consist of the newly requested document and the prefetched documents based on the value of prefetching priority determined by Priority Evaluator. All prefetched Web pages are cached in the cache directory through Proxy Caching Manager.

### 3.4 Recording Time for Implement Event

Prefetching advantage could be implemented by displaying the results of calculation of round trip time (RTT). In Fig. 3.5, we keep track of the recording time for all events happening among client, proxy, and server. We assume that a user wants to view Web page N1, which contains two hyperlinks to Web page N11 and N12. After finishing reading N1, it goes through N11, which has a hyperlink to Web page N111.

$C_i$  represents recording time on client side,  $P_i$  represents recording time on proxy side, and  $S_i$  is recording time on server side. After the proxy receives a request from the client (P1), it parses the request message for document N1 (P2). It checks the cache directory and finds that there is no cached file for N1, so it establishes a connection to the server (P3). After getting response back from the server (P4), it sends N1 back to the client (P5). Meanwhile, the proxy extracts two hyperlinks to document N11 and N12 and

prefetches them (P6)(P7) according to their priorities. The proxy receives the server's reply (P8)(P9).

At C2, the client gets N1 and begins to view it. At C3, the client sends the second request to the proxy. The difference between value of C2 and C3 is the interaction interval. On the proxy side, we call the difference between value of P5 and P10 as interaction interval.

After the proxy receives the second request from the client (P10), N11 is parsed (P11). By checking the cache directory, it realizes that document N11 has already been prefetched (P11). We call that fast prefetching (In Fig. 3.6 it is associated with slow prefetching, document N11 has not been prefetched). N11 can be immediately returned to the client (P12). Then the proxy continues to extract the hyperlink N111, which is embedded in document N11, and prefetches it from the server.

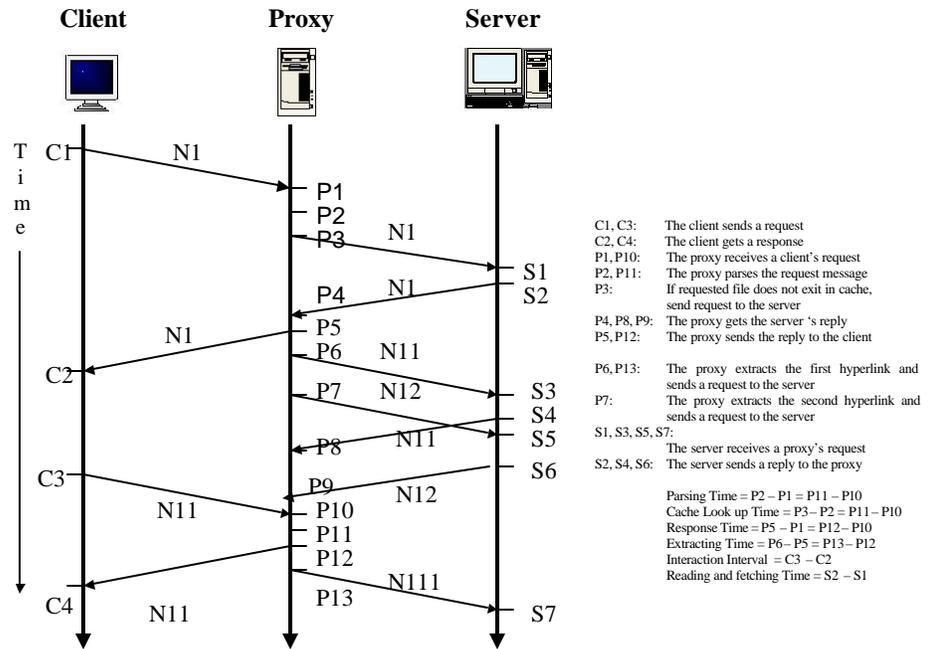


Fig. 3.5 Events Definitions and Time Distribution for Fast Prefetching

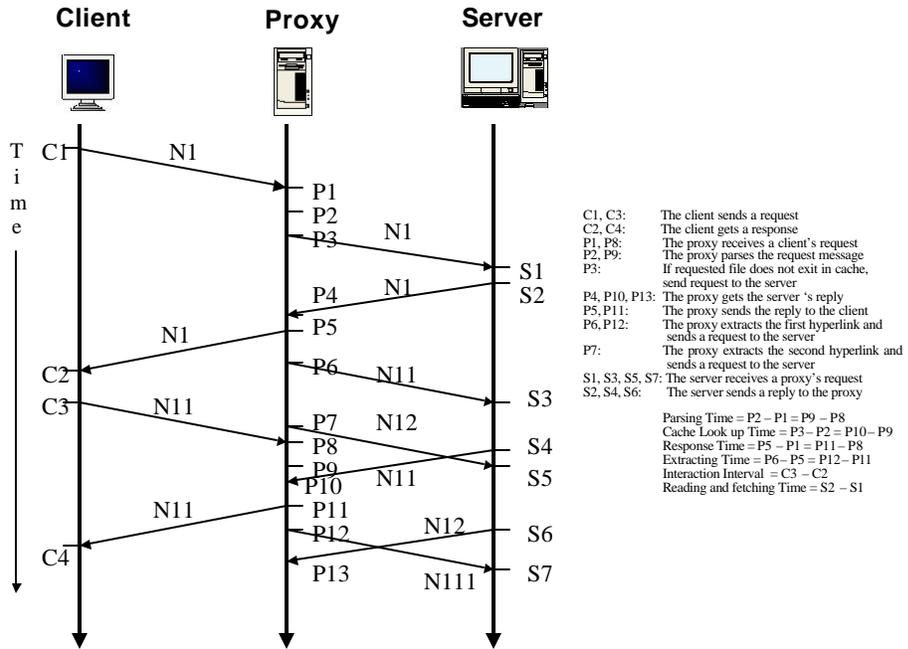


Fig. 3.6 Events Definitions and Time Distribution for Slow Prefetching

## CHAPTER 4

### Simulation Experiment

#### 4.1 Overview

Three machines were used in the simulation experiment. A 533-MHz Pentium PC with 128 MB of memory running Windows 98 was used as the proxy RHDOS. A 933-MHz Pentium PC with 128 MB of memory running Windows 98 was used as the client. Both machines connect to network with Ethernet cards.

We adopt Netscape 6 as a client browser. It should be first configured to connect to the proxy IP address. The client's cache also accepts files that the proxy prefetches. To avoid the impact of client's cache, we always clear memory cache and disk cache from Netscape before starting any test.

We created some Web pages and put them on a Web server. Each file has a size of 100 KB. Web documents are organized into four different categories. An embedded hyperlink consists of URL, the loading frequency (F), and the estimated loading time(T). It is formatted as  $T * F * URL$ .

For each experiment, we separately recorded RHDOS's response time based on the different interaction interval. We chose 5 seconds, 10 seconds, 15 seconds, 20 seconds, and 25 seconds as five different groups of interaction interval. For the different experiments, reading sequences are based on our design.

## 4.2 RHDOS implement

Run RHDOS and initialize. It will display the following information on screen:

```
Initializing...
Creating Caching Manager...
Current Free Space: 180000
The server: qtao: 8080
Proxy is running....
```

If there is no cache directory available on proxy machine, a new directory will automatically be created. When it is full, the oldest file can be deleted to create space for new Web pages.

When a user enter a specific URL, for example,

<http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N5.html>. RHDOS first checks if this Web page is already stored in cache. If yes, it hits cache and directly loads this page and returns it to the user.

```
Proxy thread 1
reqUrl--http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N5.html
Getting from cache...
```

If no, RHDOS establishes a new connection to Web server and requests it again.

```
Proxy thread 1
reqUrl--http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N5.html
Connection:Socket[addr=bristi.facnet.mcs.kent.edu/131.123.46.203,port=80
,localport=2003]
```

The Web server replies to RHDOS. Meanwhile, RHDOS reads the loaded page and parses the hyperlinks. There are three hyperlinks involved in N5.html. They are listed in the order of priority below:

```
100*1000*http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N9.html
100*2000*http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N8.html
```

150\*500\*http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N6.html

RHDOS prefetches them according to their priorities and stores them in cache.

According to formula 3.1, document N8.html should be first prefetched, then N8.html and the last is N6.html.

Current Free Space: 180000

reqUrl\*\*--http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N8.html

Caching the reply...

reqUrl\*\*--http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N9.html

Caching the reply...

reqUrl\*\*--http://bristi.facnet.mcs.kent.edu/~qtao/cache2/N6.html

Caching the reply...

RHDOS records the total time spent on loading pages. Upon completion, RHDOS waits for the next request from the client.

### **4.3 Performance Results Analysis**

The objective of the following experiment is to observe three performance impact factors: interaction interval, prefetching sequence, and reading habit. Reading habit is the actual reading sequence. HTML documents were given in fixed sizes and users walk through each probable chain of anchors during the same period of time. The performance is evaluated by the responsiveness (a ratio of cumulative lag time experienced with active prefetching to that without any prefetching) and the data volume with active prefetching.

We generate a few groups of nodes. Each node stands for one Web page. They are organized in the following four different types of connection:

#### **4.3.1 Chain**

Some Web-based quizzes, slides show, and application form are examples of this

type of connection. One of its features is that one Web page only includes one hyperlink. Only one Web document needs to be prefetched each time.

In Fig. 4.1, 6 nodes are connected in a chain. N1 is the first view object. We conducted two experiments. In the first experiment, we only read N1, N2, and N3; in the second, we went through N1, N2, N3, N4, N5, and N6.

#### 1. Response Time Analysis:

The performance for response time in chain is shown in Fig. 4.2. When the reading sequence is N1, N2, and N3, the maximum improvement in RHDOS responsiveness we observed is about 1.86 times. If we view all 6 documents, its responsiveness could be improved about 4.56 times. Actually, the more documents we view, the more improvement of responsiveness performance we can acquire, since we can view all documents as prefetched except for N1. In addition, the responsiveness is not affected by interaction interval. The minimum interaction interval can guarantee that one Web document could be prefetched.

#### 2. Data Volume Analysis:

The performance for data volume in chain is shown in Fig. 4.3. When the reading sequence is N1, N2, and N3, the maximum amount of data is 4 units. Compared to the data volume without prefetching, only one extra unit data volume was increased. If we view all 6 documents, 6 units of data volume will be transferred and no extra amount of data is produced. So, whatever the reading sequence is, the maximum extra data volume is one unit.

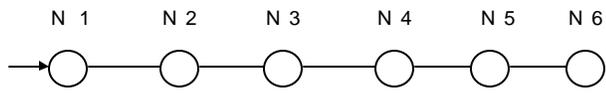


Fig. 4.1 A Chain

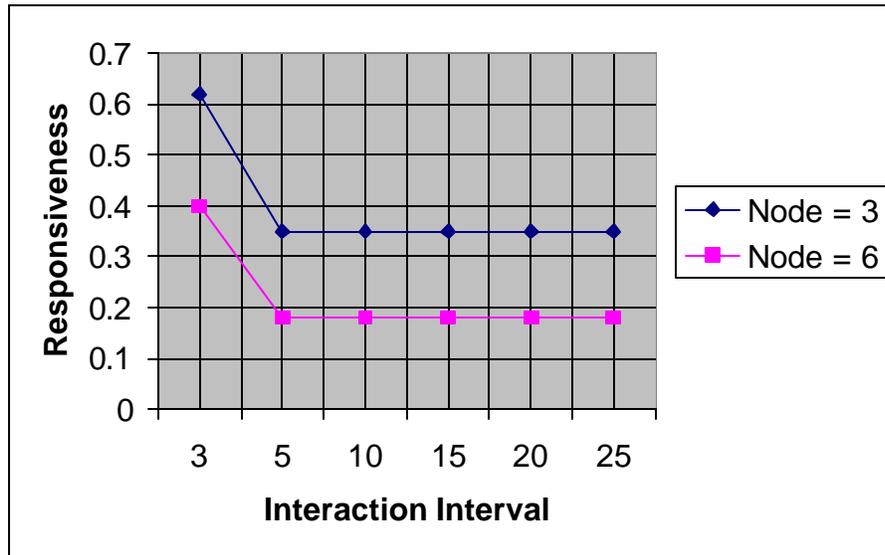


Fig. 4.2 Performance for Response Time in Chain

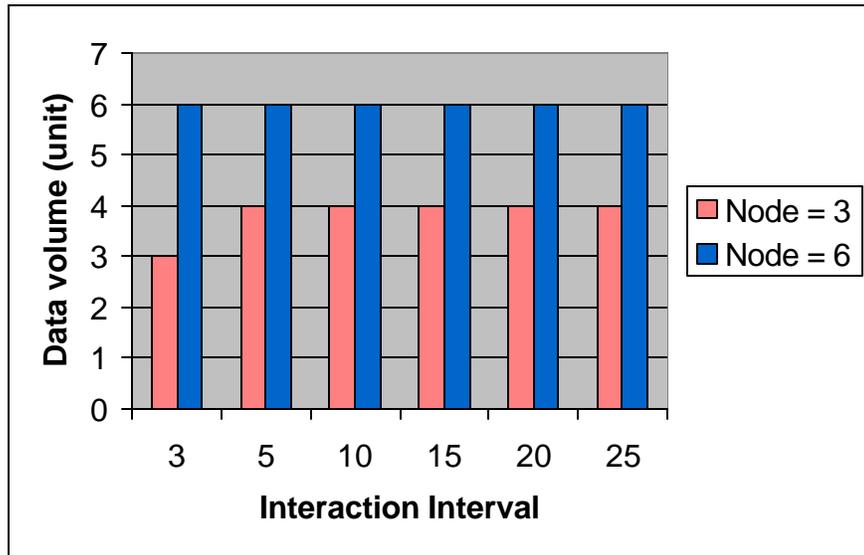


Fig. 4.3 Performance for Data Volume in Chain

### **4.3.2 Tree**

Some items are summarized into different categories on a Web page. We consider it as a tree organization. Each Web page includes its own hyperlinks or child pages. Meanwhile, it is either a direct or indirect child page of the main page. The tree organization is characterized by the fact that each page could have many child pages, but only one parent page. In the following text we will analyze both organization situations: a full tree and a path in a tree.

#### **4.3.2.1 A Full Tree**

In Fig. 4.4, 31 nodes are organized into a tree with three levels (height equals 3). Each of N0, N1, N2, N3, N4, and N5 contains five hyperlinks. The branch factor equals 5. In Fig. 4.5, height also equals 3, but branch factor is only 3, since each of N0, N1, N2, and N3 contains three hyperlinks. Their prefetching sequences are classified into two different types in table 4.1. Reading sequences are classified into three different types in table 4.2.

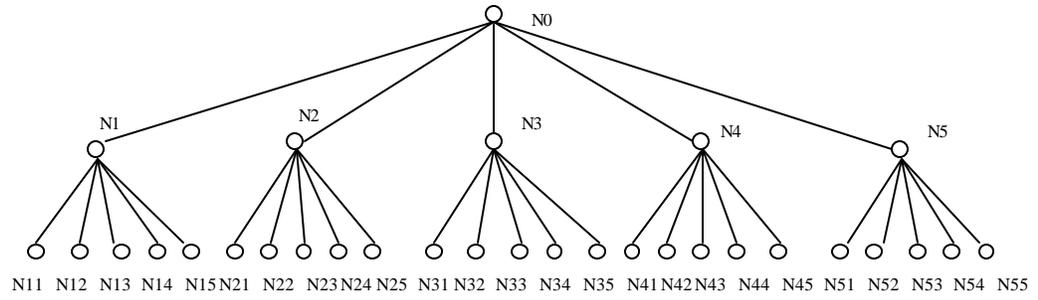


Fig. 4.4 A Full Tree [ $H = 3$ ,  $BF = 5$ ]

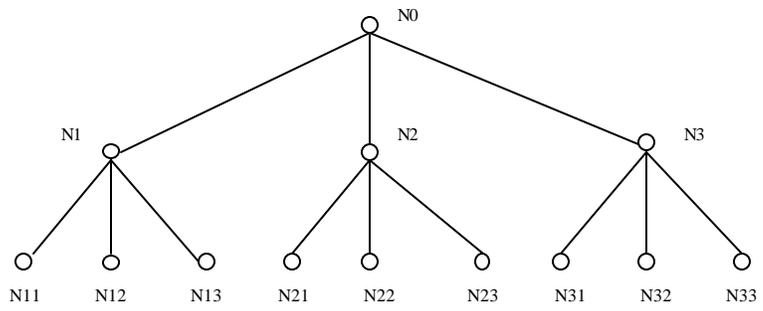


Fig. 4.5 A Full Tree [H = 3, BF = 3]

| Tree Type     | Node | Prefetching Sequence |                     |
|---------------|------|----------------------|---------------------|
|               |      | Left First           | Right First         |
| H = 3, BF = 5 | N0   | N1,N2,N3,N4,N5       | N5,N4,N3,N2,N1      |
|               | N1   | N11,N12,N13,N14,N15  | N15,N14,N13,N12,N11 |
|               | N2   | N21,N22,N23,N24,N25  | N25,N24,N23,N22,N21 |
|               | N3   | N31,N32,N33,N34,N35  | N35,N34,N33,N32,N31 |
|               | N4   | N41,N42,N43,N44,N45  | N45,N44,N43,N42,N41 |
|               | N5   | N51,N52,N53,N54,N55  | N55,N54,N53,N52,N51 |
| H = 3, BF = 3 | N0   | N1,N2,N3             | N3,N2,N1            |
|               | N1   | N11,N12,N13          | N13,N12,N11         |
|               | N2   | N21,N22,N23          | N23,N22,N21         |
|               | N3   | N31,N32,N33          | N33,N32,N31         |

Table 4.1 Lists of Prefetching Sequences in a Full Tree

| Tree Type       | Reading Sequence  |   |   |
|-----------------|---|---|---|
|                 | Depth First   | Breadth First   | Random  |
| H = 3<br>BF = 5 | N0, N1, N11, N12,<br>N13, N14, N15, N2,<br>N21, N22, N23,<br>N24, N25, N3, N31,<br>N32, N33, N34,<br>N35, N4, N41, N42,<br>N43, N44, N45, N5,<br>N51, N52, N53,<br>N54, N55 | N0, N1, N2, N3, N4,<br>N5, N11, N12, N13,<br>N14, N15, N21,<br>N22, N23, N24,<br>N25, N31, N32,<br>N33, N34, N35,<br>N41, N42, N43,<br>N44, N45, N51,<br>N52, N53, N54, N55 | N0, N4, N41, N42,<br>N2, N21, N22, N23,<br>N24, N25, N3, N33,<br>N31, N32, N34,<br>N35, N1, N5, N52,<br>N53, N54, N55,<br>N51, N43, N44,<br>N45, N11, N12,<br>N13, N14, N15 |
| H = 3<br>BF = 3 | N0, N1, N11, N12,<br>N13, N2, N21,<br>N22, N23, N3, N31,<br>N32, N33  | N0, N1, N2, N3,<br>N11, N12, N13,<br>N21, N22, N23,<br>N31, N32, N33  | N0, N1, N2, N3,<br>N11, N12, N13,<br>N21, N22, N23,<br>N31, N32, N33  |

Table 4.2 Lists of Reading Sequences in a Full Tree

## 1. Response Time Analysis:

The performance for response time in a full tree with Left First and Right First as prefetching sequence are shown in Fig. 4.6 and Fig. 4.7 respectively.

We observe that the prefetching sequence and reading sequence affect the performance for response time. The improvement in RHDOS responsiveness is the best when we compare reading Web documents in Depth First with in Breadth First and Random. In Fig. 4.6, when prefetching sequence is Left First, The responsiveness with Random and Breadth First is up to 2.4 and 3.7 times less than that with Depth First respectively. In Fig. 4.7, when prefetching sequence is Right First, the responsiveness with Random and Breadth First is up to 0.6 and 0.7 times less than that with Depth First respectively.

No matter what the prefetching sequence is, with the number of branching factor increasing, the impact of prefetching performance always increases. In addition, with growing interaction interval, the value of responsiveness decreases gradually. The reason is the more interaction interval there is, the more efficiently prefetching is implemented. So more improvements of performance are acquired.

## 2. Data Volume Analysis:

The performance for data volume in a full tree is shown in Fig. 4.8. Whatever the branching factor is, data volume is not affected by prefetching sequence and reading sequence. Interaction interval even does not affect its performance for data volume. The total amount of transferring data is the same as without prefetching. When the branching

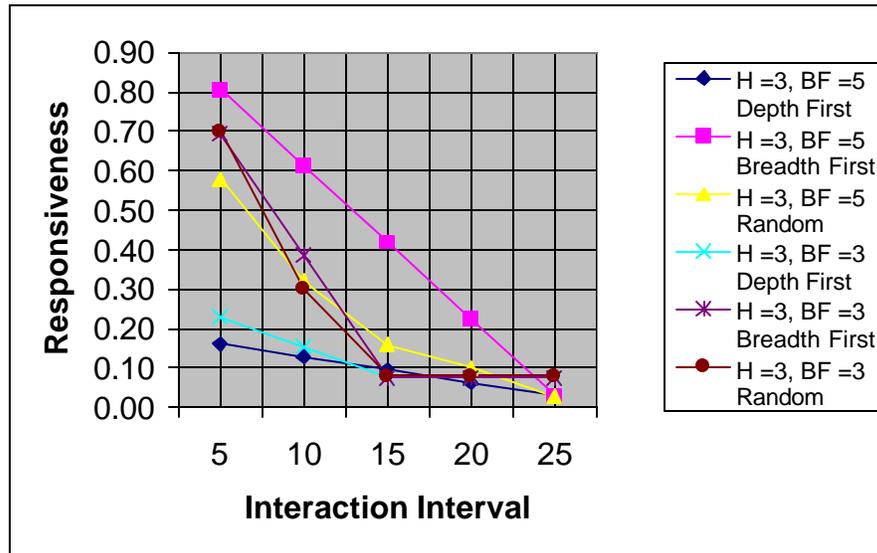


Fig. 4.6 Performance for Response Time in Tree with Left First

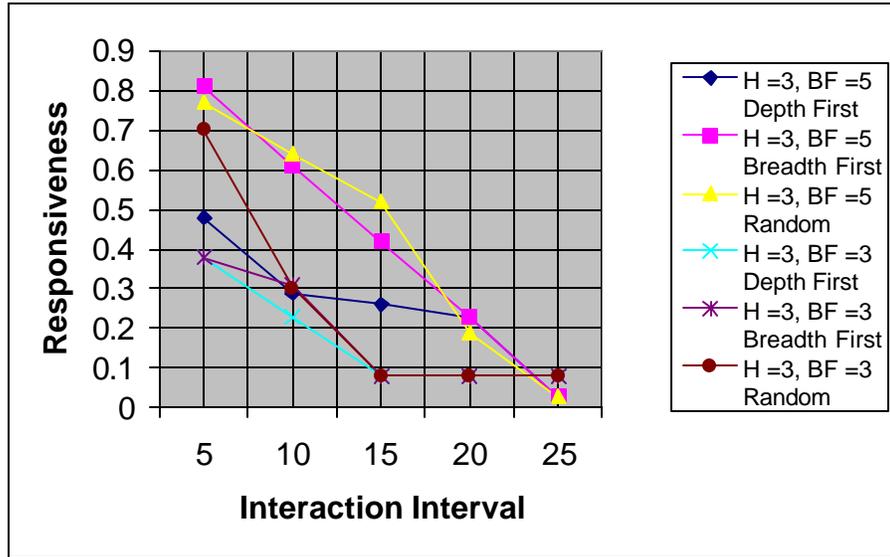


Fig. 4.7 Performance for Response Time in Tree with Right First

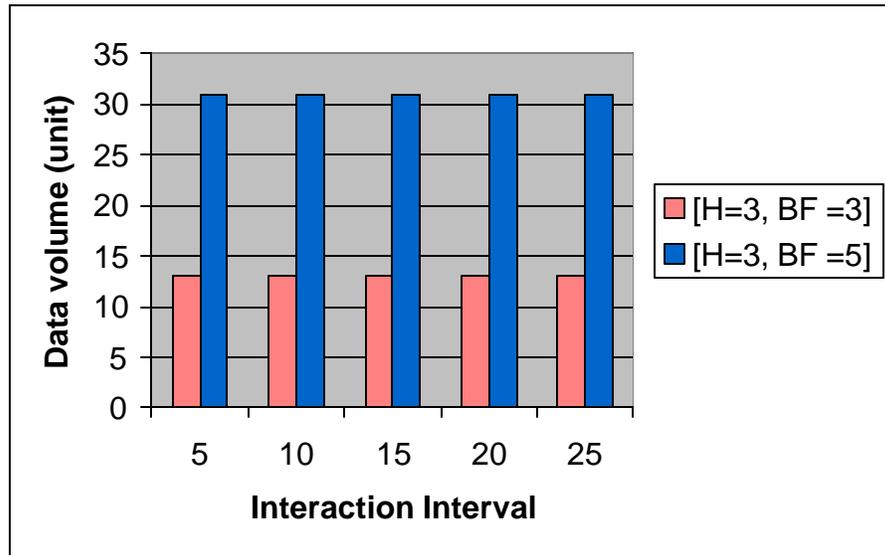


Fig. 4.8 Performance for Data Volume in A Full Tree

factor is 5, data volume is 31 units; when the branching factor is 3, data volume is 13 units.

#### **4.3.2.2 A Path in a Tree**

In Fig. 4.9, we only went through some of Web documents. We chose three parts of nodes to view separately. In path 1, we moved through N0, N1, N11, N111, and N1111 in order; in path 2, the reading sequence is N0, N1, N12, N122, and N1221; Path 3 is N0, N2, N22, N222, and N2222.

Experiment is based on two different prefetching sequences: Left First and Right First. We adopt the same implementation methods as in the experiment with a full tree.

##### **1. Response Time Analysis:**

The performance for response time in one path in a tree reading is shown in Fig. 4.10. We observe that path 1's responsiveness with Left First prefetching sequence is the same as path 3's one with Right First prefetching sequence.

We can also find that part 3's responsiveness with Left First prefetching sequence is the same as part 1's one with Right First prefetching sequence. If interaction interval is 5 seconds, the response time with prefetching is the same as that without prefetching, since the next page we will move through is not a prefetched document.

However, whatever prefetching sequence is, either Left First or Right First, part 2 always has the same change for the responsiveness value.

When prefetching sequence is Left First, the prefetching performance in path 1 is

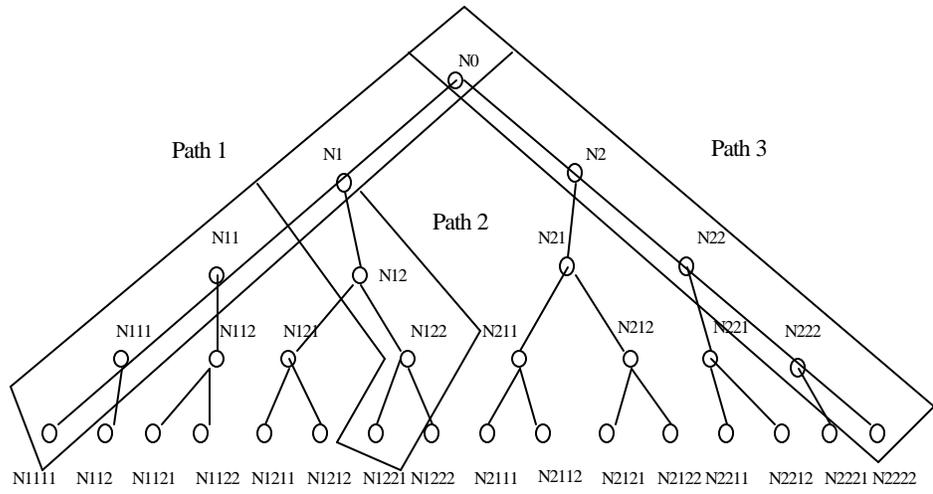


Fig. 4.9 Paths in a Tree

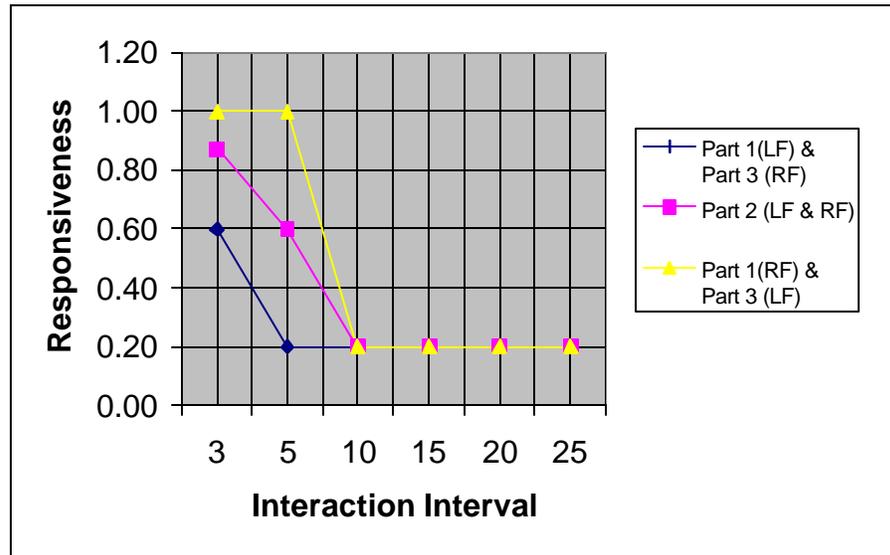


Fig. 4.10 Performance for Response Time in Paths of a Tree

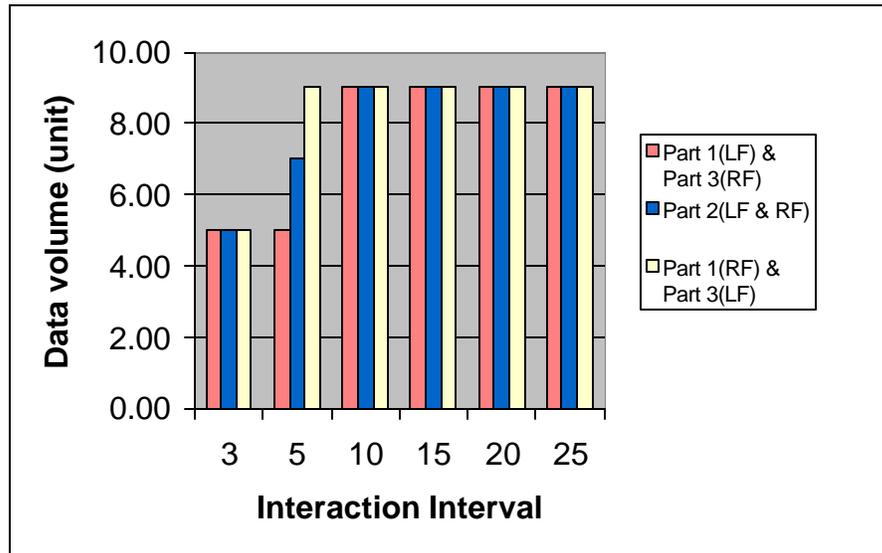


Fig. 4.11 Performance for Data Volume in Paths of a Tree

better than that in path 2 and path 3. The responsiveness with path 2 and path 3 is up to 2 and 4 times less than that with part 1 respectively.

With growing interaction interval, the system responsiveness always increases in a gradual fashion for path 1, path2, and path 3.

## 2. Data Volume Analysis:

The performance for data volume in one path in a tree reading is shown in Fig. 4.11. If interaction interval is 5 seconds, path 1's data volume with Left First prefetching sequence is 5 units, same as the path 3's one with Right First prefetching sequence. The path 3's data volume with Left First prefetching sequence is 9 units, same as the part 1's one with Right First prefetching sequence. No matter what prefetching sequence path 2 uses, its data volume is 7 units. With Left First prefetching sequence, the amount of unnecessary data in path 2 and path 3 is up to 40% and 80% more than that in path 1 respectively.

Once it reaches 10 seconds, the performance for data volume in part 1, part 2, and part 3 are the same. They are all 9 units no matter what their prefetching sequences are.

### 4.3.3 Fully Connected graph

Most online encyclopedia and e-books fall into this category of organization. The hyperlinks for chapters and sections of each e-book are usually fully connected. We organized two different situations with 6 and 10 nodes separately. In Fig. 4.12, each node contains 5 hyperlinks. In Fig. 4.13, each node contains 9 hyperlinks. They are fully connected with each other. In table 4.3, we suppose the prefetching sequence is in

clockwise. In table 4.4, reading sequences are divided into three different types such as Clockwise, Counter clockwise, and Random.

#### 1. Response Time Analysis:

The performance for response time in fully connected graph is shown in Fig. 4.14. No matter how many nodes they have, the prefetching performance in clockwise reading direction is always better than that in counter clockwise. The prefetching performance in random reading direction is in between clockwise and in counter clockwise directions. The responsiveness with Random and Counter Clockwise is up to 5.3 and 10.3 times less than that with CW respectively. With growing number of nodes, the impact of prefetching performance increases. With growing interaction interval, the system responsiveness increases in a gradual fashion.

#### 2. Data Volume Analysis:

The performance for data volume in fully connected graph is shown in Fig. 4.15. Different reading sequences result in different performance of data volume. The amount of data in clockwise reading direction is always less than that in counter clockwise. Basically data volume for any reading order always increases gradually when interaction interval increases gradually. All of them produce a lot of extra amount of data compared to amount of transferred data without prefetching. The more nodes we move through, the more extra amount of data is produced.

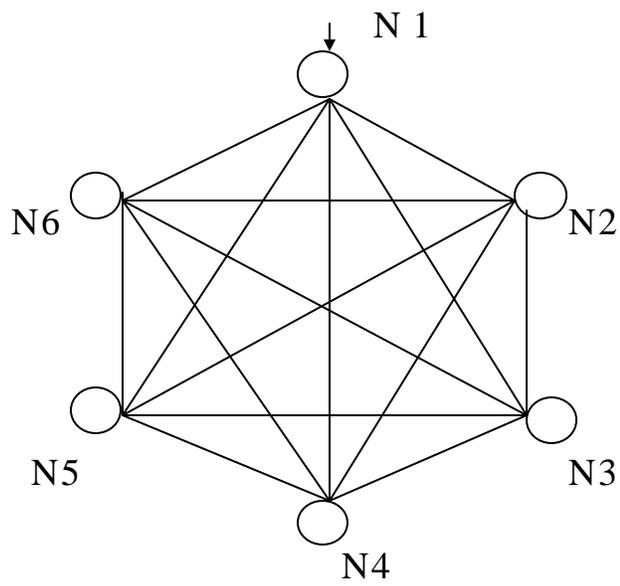


Fig. 4.12 A Fully connected Graph with 6 Nodes

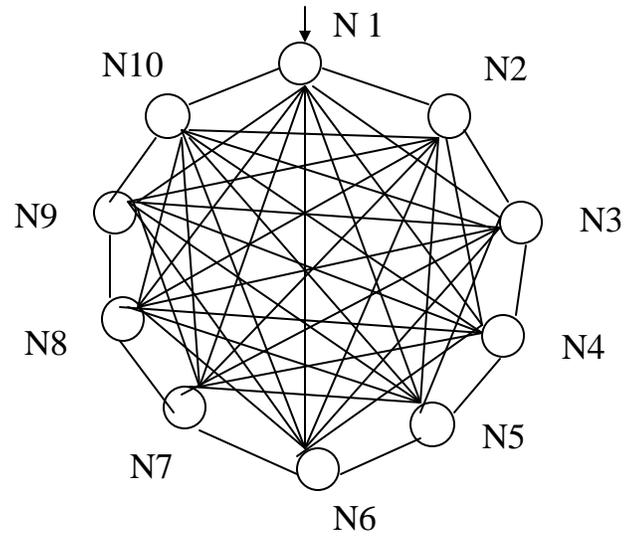


Fig. 4.13 A Fully connected Graph with 10 Nodes

| Total Nodes | Node | Prefetching Sequence        |
|-------------|------|-----------------------------|
|             |      | Clockwise                   |
| 6           | N1   | N2,N3,N4,N5,N6              |
|             | N2   | N3,N4,N5,N6,N1              |
|             | N3   | N4,N5,N6,N1,N2              |
|             | N4   | N5,N6,N1,N2,N3              |
|             | N5   | N6,N1,N2,N3,N4              |
|             | N6   | N1,N2,N3,N4,N5              |
| 10          | N1   | N2,N3,N4,N5,N6,N7,N8,N9,N10 |
|             | N2   | N3,N4,N5,N6,N7,N8,N9,N10,N1 |
|             | N3   | N4,N5,N6,N7,N8,N9,N10,N1,N2 |
|             | N4   | N5,N6,N7,N8,N9,N10,N1,N2,N3 |
|             | N5   | N6,N7,N8,N9,N10,N1,N2,N3,N4 |
|             | N6   | N7,N8,N9,N10,N1,N2,N3,N4,N5 |
|             | N7   | N8,N9,N10,N1,N2,N3,N4,N5,N6 |
|             | N8   | N9,N10,N1,N2,N3,N4,N5,N6,N7 |
|             | N9   | N10,N1,N2,N3,N4,N5,N6,N7,N8 |
|             | N10  | N1,N2,N3,N4,N5,N6,N7,N8,N9  |

Table 4.3 Lists of Prefetching Sequences in a Fully Connected Graph

| Total<br>Nodes | Reading Sequence                   |                                    |                                    |
|----------------|------------------------------------|------------------------------------|------------------------------------|
|                | Clockwise                          | Counter Clockwise                  | Random                             |
| 6              | N1,N2,N3,N4,N5,N6                  | N1,N6,N5,N4,N3,N2                  | N1,N4,N6,N2,N5,N3                  |
| 10             | N1,N2,N3,N4,N5,N6<br>,N7,N8,N9,N10 | N1,N10,N9,N8,N7,N6,<br>N5,N4,N3,N2 | N1,N6,N3,N5,N9,N7,N2,<br>N8,N4,N10 |

Table 4.4 Lists of Reading Sequences in a Fully Connected Graph

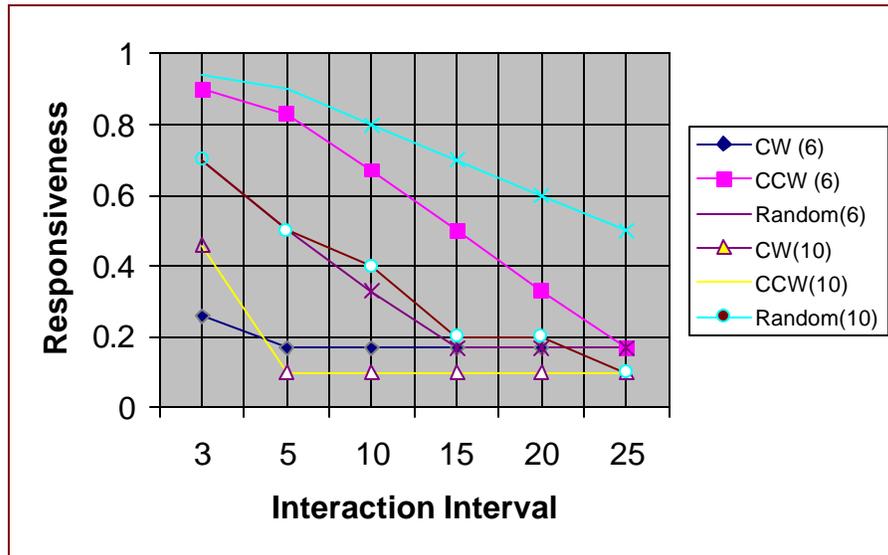


Fig. 4.14 Performance for Response Time in Fully Connected Graph

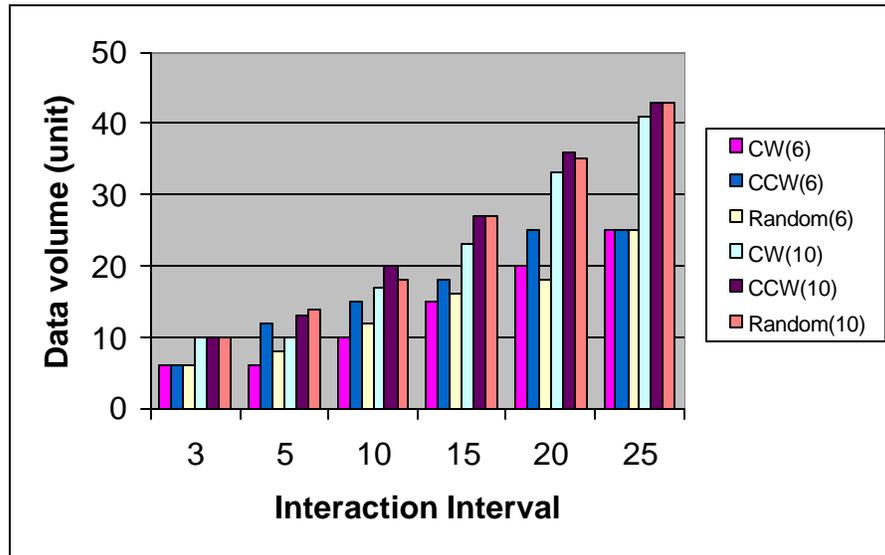


Fig. 4.15 Performance for Data Volume in Fully Connected Graph

#### 4.3.4 Tree with Core Graph

This type of organization is quite popular for Web pages. If there is a menu on a Web page and the page is created using frameset, it is often this type of organization.

In Fig. 4.16, there are two cores. One core is consisted of N0, N1, N2, and N3, we refer to it as core 1. Another core is consisted of N4, N5, and N6, we refer to it as core 2. Each node is a parent in the core. It has its own children. For instance, N0 is a member of core 1. Meanwhile, it is the parent of three children, N4, N5, and N6, which are members of core 2.

Core Set means all members of the core are fully connected. Child Set is a tree structure. Two types of prefetching sequences are selected. We call them Core Set First and Child Set First. See table 4.5.

##### 1. Response Time Analysis:

The performance for response time in Tree with Core is shown in Fig. 4.17.

With interaction interval increased, the value of responsiveness decreases gradually for both Core Set First and Child Set First. However, if we use Child Set First as prefetching sequence, its performance improvement for responsiveness is better than Core Set First. That means Child Set First closely matches the Depth First reading sequence. The responsiveness with Core Set First is up to 2 times less than that with Child Set First.

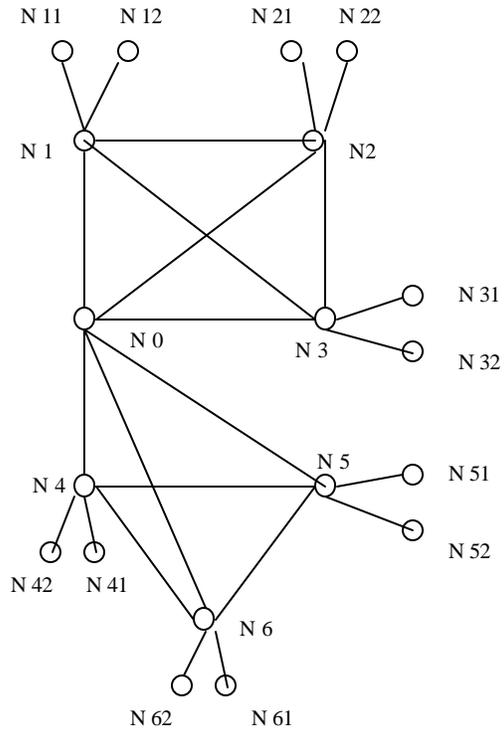


Fig. 4.16 A Tree with Core

| Node | Prefetching Sequence |                   | Reading Sequence<br>(Depth First)   |
|------|----------------------|-------------------|---|
|      | Child Set First      | Core Set First    |   |
| N0   | N1,N2,N3,N5,N4,N6    | N1,N2,N3,N5,N4,N6 | N0,N4,N41,N42,N6,N61,<br>N62,N5,N51,N52,N1,N11,<br>N12,N2,N21,N22,N3,N31, N32 |
| N1   | N11,N12,N2,N3,N0     | N2,N3,N0,N11,N12  |   |
| N2   | N21,N22,N3,N0,N1     | N3,N0,N1,N21,N22  |   |
| N3   | N31,N32,N0,N1,N2     | N0,N1,N2,N31,N32  |   |
| N4   | N41,N42,N0,N5,N6     | N5,N6,N41,N42,N0  |   |
| N5   | N51,N52,N0,N6,N4     | N6,N4,N51,N52,N0  |   |
| N6   | N61,N62,N0,N4,N5     | N4,N5,N61,N62,N0  |   |

Table 4.5 Lists of Sequences in a Tree with Core Graph

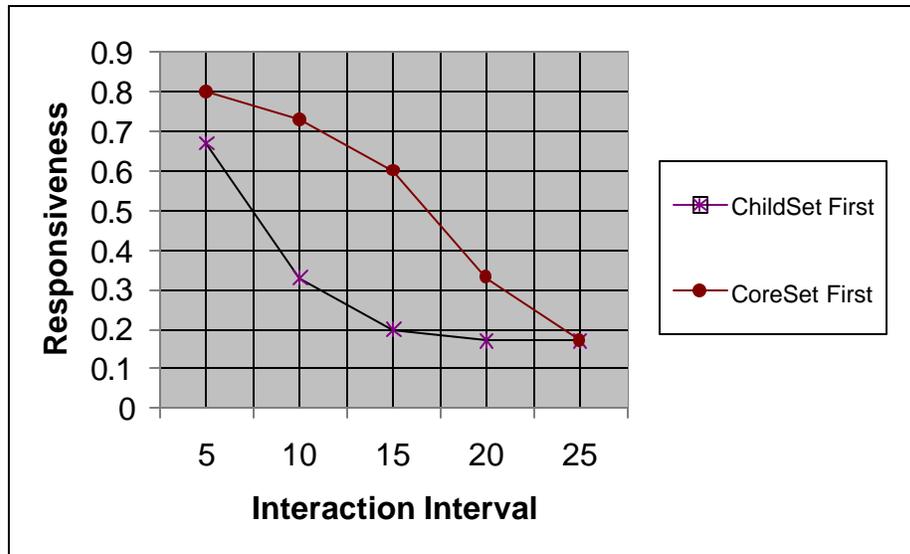


Fig. 4.17 Performance for Response Time in a Tree with Core Graph

## 2. Data Volume Analysis:

The performance for response time in Tree with Core is shown in Fig. 4.18. If Child Set First is selected as prefetching sequence, its performance improvement for data volume is better than Core Set First. The amount of unnecessary data with Core Set First is up to 43% more than that with Child Set First.

With interaction interval increased, the data volume increases gradually for both of them, and the extra amount of data also increase gradually.

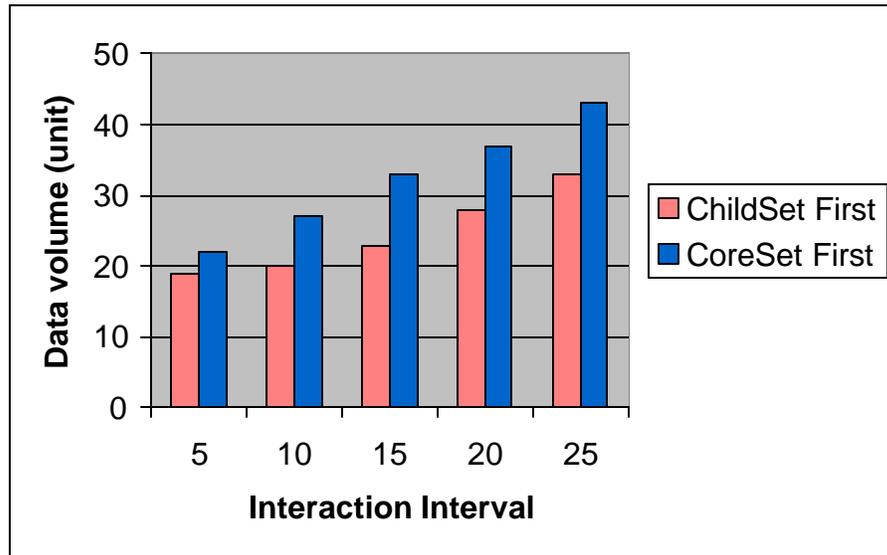


Fig. 4.18 Performance for Data Volume in a Tree with Core Graph

## CHAPTER 5

### Conclusions and Future Works

#### 5.1 Conclusions

The effectiveness of prefetching is particularly significant for Internet reference. The prefetching prediction model reduces access lag for new references. With the current research level, any prefetching is a kind of gambling. The primary difficulty in all prefetching mechanisms is to be able to accurately predict which pages will be needed next, to minimize mistakes that result in wasted bandwidth and increased server loads.

A HTTP proxy system is developed that simulates the performance of the prefetching technique. Experiment is based on four different types of Web documents organization such as chain, tree, fully connected graph, and complex tree with core. Three performance impact factors we selected include interaction interval, prefetching sequence, and reading sequence.

Analysis results show that, compared to a matched system, the response time of a random system can take 1.6 - 6.3 times larger and bring in 1.8 - 2.0 times more unnecessary data. In the worst case, a completely mismatched system's response time can be about 1.7 - 11.3 times larger and result in 1.3 - 1.4 times more unnecessary data than a matched system. Smarter prefetching techniques can be developed if we take the

structure of webspace and user reading behavior into consideration. This study may also help content developer organize the webspace so that it can be navigated faster.

## **5.2 Future Works**

As of further research work, we think the following directions are very interesting:

RHDOS is only experiment system. The result indicates that for effective prefetch more knowledge about the user reading behavior and document organization can be beneficial. Future work should be to design mechanism to obtain these information. For example, we need to continue solving the problem how the proxy gets real frequency and estimated loading time for the Web pages requested by a client.

Currently there is no technique to acquire a hyperspace pattern. The pattern information is distributed into multiple pages. HTTP and other existing mechanism cannot help with discovering or expressing the hidden pattern. More research is required on such mechanism. Perhaps an XML extension or HTTP can be an interesting topic for future study.

In order to further investigate reader habit, we should try to track some Web servers. It will be useful to track which Web pages were requested from different persons in a given time period.

To further do research on whether the user-agent device including the Web server, the caching proxy, and the browser can impact on the interaction time.

Study the impact of proxy cache size, media component classification, and discarding policies. We suspect reading time will show high correlation with media type

and even content. Additional study can be performed to determine the extents.

## References

- [CoKa00] E. Cohen and H. Kaplan. Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency. Proc. of the IEEE INFOCOM 2000, Tel-Aviv, Israel, March 2000.
- [CrBa98] M. Crovella, P. Barford, The Network Effects of prefetching, Proc. Of IEEE INFOCOM 1998, San Francisco, USA, 1998.
- [Davi01] Brian D. Davison. Assertion: Prefetching With GET Is Not Good. Proc. Of the 6<sup>th</sup> International Workshop on Web Caching and Content Distribution, June 20-22, 2001.
- [Duch99] D. Duchamp. Prefetching Hyperlinks. Proceedings of the USENIX Symposium on Internet Technologies and Systems, Colorado, USA, October 1999.
- [[Http://www.usenix.org/events/usits99](http://www.usenix.org/events/usits99)].
- [FMFM99] R. Fielding, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol-HTTP/1.1. Tech. Rep. RFC 2616 (June), IETF. 1999.
- [<http://www.ietf.org/rfc/rfc2616.txt>]
- [FIMD97] Todd B. Fleming, Scott F. Midkiff, and Nathaniel J. Davis, IV. Improving the Performance of the World Wide Web over Wireless Networks. Globecom'97.
- [Retrieved from: [http://www.cs.columbia.edu/~hgs/InternetTC/GlobalInternet97/Flem9711\\_Improving.pdf](http://www.cs.columbia.edu/~hgs/InternetTC/GlobalInternet97/Flem9711_Improving.pdf)]
- [JaCa98] Q. Jacobson, Pei Cao, Potential and Limits of Web Prefetching Between Low-Bandwidth Clients and Proxies, 3rd International WWW Caching Workshop,

Manchester, England, June 15-17 1998.

[Khan99] Javed I. Khan, Ordering Prefetch in Trees, Sequences and Graphs, Technical Report 1999-12-03, Kent State University, [available at URL <http://medianet.kent.edu/technicalreports.html>, also mirrored at <http://bristi.facnet.mcs.kent.edu/~javed/medianet>.

[Khan00] Javed I. Khan, Active Streaming in Transport Delay Minimization, Workshop on Scalable Web Services, Toronto, pp95-102, August 2000.

[KhTa01] Javed I. Khan, Qingping Tao, Partial Prefetch for Faster Surfing in Composite Hypermedia, the 3rd USENIX Symposium on Internet Technologies USITS'01, San Francisco, pp13-24, March 2001.

[KhTa01] Javed I. Khan, Qingping Tao, Prefetch scheduling for composite hypermedia, Proceedings of IEEE International Conference on Communications (ICC2001), Finland, pp 768-773, June 2001.

[KaPJ99] M. Frans Kaashoek, Tom Pinckney, and Joshua A. Tauber, Dynamic Documents: Extensibility and Adaptability in the WWW, <http://www.pdos.lcs.mit.edu/papers/www94.html>.

[KrLM97] T. Kroeger, D. D. E. Long & J. Mogul, Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, Proc. of USENIX Symposium on Internet Technology and Systems, Monterey, December 1997, pp-319-328.

[PaMe99] T. Palpanas and A. Mendelzon, Web Prefetching Using Partial Match Prediction, WWW Caching Workshop, San Diego, CA, March 1999.

[PiPi99] P. Pirolli and J. E. Pitkow, Distributions of surfers' paths through the World Wide Web: Empirical characterizations, Journal of World Wide Web, v.1-2, pp29-45, 1999.

## Appendix

### Lists of Some Examples for Web pages Organization

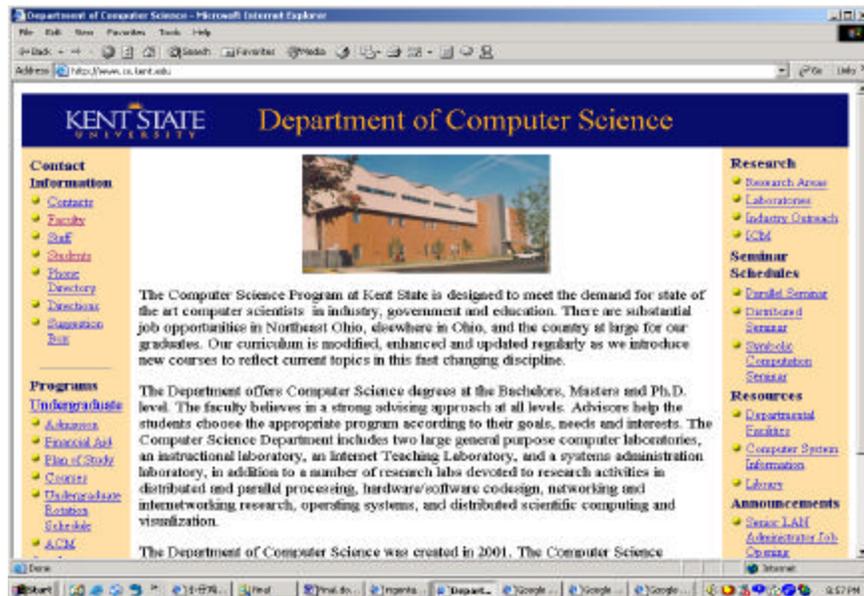
The screenshot shows a web browser window titled "ingenta: personal registration - Microsoft Internet Explorer". The address bar displays the URL: <http://www.ingenta.com/isis/register/RegisterPersonalUser/ingenta>. The page content includes a registration form with the following elements:

- Academic disciplines selection:
  - engineering/technology
  - arts and humanities
  - nursing
  - philosophy/linguistics
  - agriculture/food sciences
  - politics
- Registration fields:
  - \* choose a user name:
  - \* choose a password: (must be 5 characters or more)
  - \* confirm password:
- Subscription options:

If you are registering from an institution or a company make sure you check the box to gain free full-text access to your institution's/company's subscriptions whether you are on or off site.

  - I would like full-text access to my institution's/company's subscriptions
  - Keep up-to-date with new titles available and new features and functionality on ingenta.com by signing up to receive Ingenta Announce.
- Navigation: "Now, register your subscriptions" with a "continue" button.

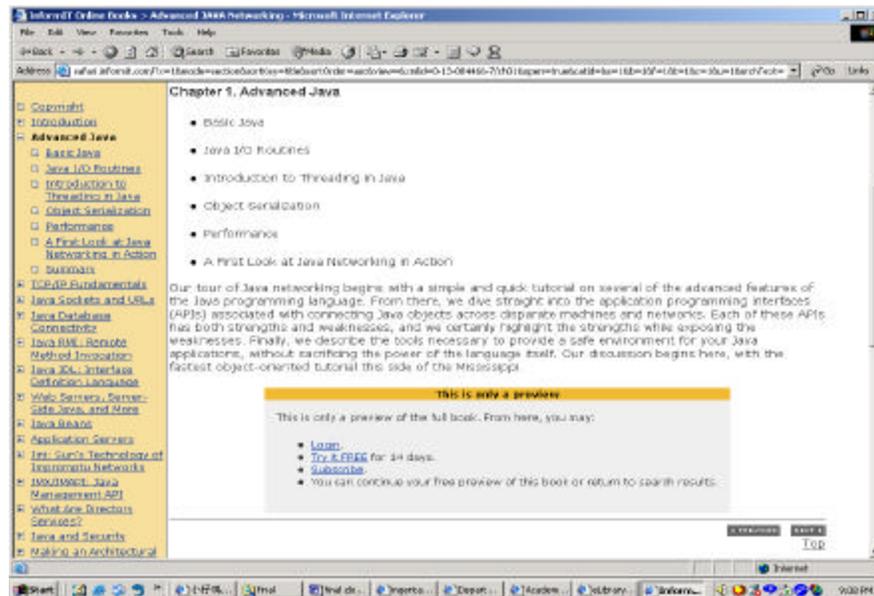
<http://www.ingenta.com/isis/register/RegisterPersonalUser/ingenta>



<http://www.cs.kent.edu>



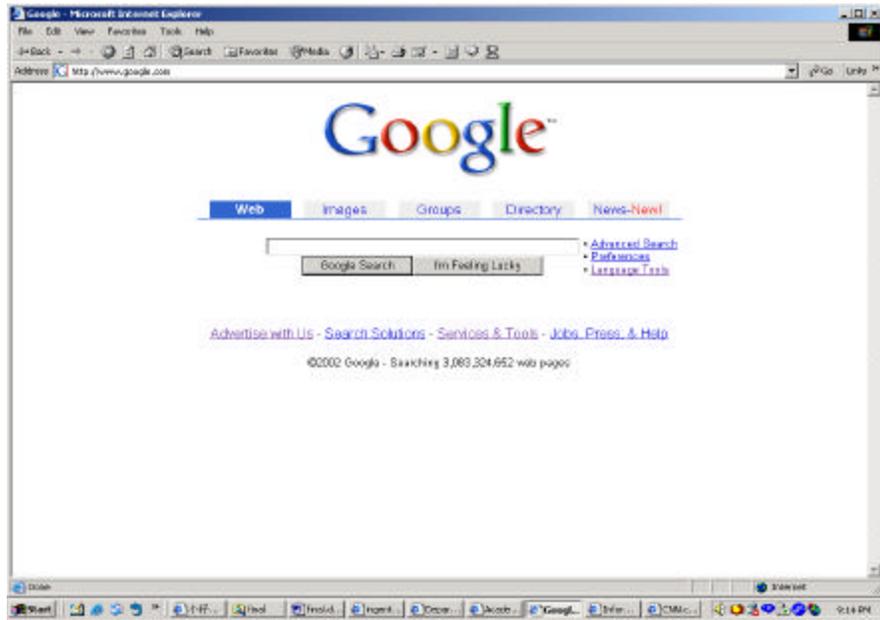
<http://www.kent.edu/academics/>



<http://safari.informit.com/?XmlId=0-13-084466-7>



<http://www.cnn.com/SHOWBIZ/>



<http://www.google.com>