MODELING AND ACCELERATION OF CONTENT DELIVERY IN WORLD WIDE WEB

YUAN JUNLI

NATIONAL UNIVERSITY OF SINGAPORE

2005

MODELING AND ACCELERATION OF CONTENT DELIVERY IN WORLD WIDE WEB

YUAN JUNLI
(M.Eng. USTC, B.Eng. JUT, PRC)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2005

Acknowledgements

First of all, I would like to take this opportunity to express my heartfelt thanks to my supervisor, Prof. Chi Chi-Hung, for his invaluable advice, assistance and encouragement throughout the course of my study. I benefited tremendously from his guidance and insights in this field. He also spent a lot of time and effort coaching me on thesis writing. Besides his help on my research work, he is also an invaluable mentor of my life. His spirit will inspire and benefit me in the rest of my life. I could not thank him enough and I hope I will have chance to continue working with him.

I am indebted to Dr. Sun Qibin for his kind and generous help on my thesis writing. Without his help, this work would not be finished smoothly.

In the course of my study, many other people have helped me in one way or another. I would like to thank Mr. Jerry Hoe, Dr. Feng Huaming, Dr. Li Xiang, Dr. Zhao Yunlong, Dr. Ding Chen and Dr. Lin Weidong for their discussions, suggestions and encouragements. I also very much enjoyed working with the talented fellow students in MMI lab where I did my Ph.D. study: Deng Jing, Lim Ser Nam, Lu Sifei, Wang Hongguang, Henry Novianus Palit, William Ku, Chua Choon Keng, Su Mu, Ting Meng Yean, Zhang Shutao and Zhang Luwei etc. Besides their helpful discussion and cooperation on my research, their friendship and support also made my work and life very enjoyable over the years.

I would also like to thank the National University of Singapore for providing me the research scholarship. I am also grateful to the School of Computing for providing an excellent environment for study and research.

Last but not least, many thanks go to my parents, my wife and all other family members for their understanding and support during the long course of my studies. Without their constant loving support, this work would not exist.

Table of Contents

List of Figures

List of Tables

Table of Abbreviations

Abbreviation	Description
CAGSC	Content-Aware Global Static Compression
CLP	Chunk Level Parallelism
CO	Container Object
CST	Chunk Sequence Time
CT	Connection Time
DT	Definition Time
EO	Embedded Object
EOD	Embedded Object Declaration
EOIP	Embedded Object Information Propagation
ET	Ending Time
LRT	Location Resolution Time
NLANR	National Laboratory for Applied Network Research
OFL	Object Fetch Latency
ORL	Object Retrieval Latency
RST	Request Sending Time
RTT	Round Trip Time
SLP	Server Location Propagation
TTL	Time To Live
URI	Uniform Resource Identifiers
URL	Uniform Resource Locators
WRDG	Web Retrieval Dependency Graph
WRDM	Web Retrieval Dependency Model
WT	Waiting Time

Summary

With the explosive growth of the web, web retrieval latency has become one of the principal concerns to most web users and web content providers. Although many works have been done to understand and improve web retrieval performance, there are still some open issues in this area. In previous studies, page retrieval latency is not given enough attention; most existing studies are based on object level information, which is insufficient and sometimes even inaccurate. Also, the details of web retrieval at operation and chunk level are not well studied and understood. Furthermore, we still lack of a precise model for capturing and studying web retrieval performance. Finally, there still lack of effective acceleration mechanisms with special emphasis on improving page retrieval latency.

This thesis tackles the above issues in the area of modeling and acceleration of web content delivery. In our studies, we first examined and tried to improve the performance of the traditional way of web acceleration, i.e. web caching, by studying the effectiveness of cacheability factors in the multi-factor co-occurrence situation and the accuracy of the settings for the TTLs of web objects. Then we proposed a fine grained Web Retrieval Dependency Model (WRDM) to provide more precise capture of web retrieval process. Based on the model, we profoundly studied the factors in web retrieval process at various levels, including the detailed operation and chunk level, and page level. The results shed light on the details of object retrieval latency and the complicated relationship between object latency and page latency. It revealed that the actual object fetch latency is often less of a problem for web retrieval than the Definition Times and the Waiting Times when page latency is concerned. We also analyzed the possible impact of real-time content transformation on web retrieval latency and derive various

upper bounds for web acceleration, which revealed some low-level impacts of real-time content transformation and potentials of web acceleration.

With the guidance of the WRDM model, we systematically analyzed the effect of an important acceleration mechanism, namely web compression. The detailed analysis revealed some important effects and implication of compression on page retrieval latency. Realizing the deficiencies in general-purpose compression algorithms in the specific area of web content delivery, we proposed a new compression mechanism, named Content-Aware Global Static Compression (CAGSC), to improve the performance of compression in web content delivery.

Based on the findings from the studies using the WRDM model, we proposed some new ways to web acceleration. Besides the novel compression mechanism mentioned above, we also proposed and studied innovative acceleration mechanisms in two aspects: the dependency related mechanisms which are the Server Location Propagation mechanism (SLP) and Embedded Object Information Propagation mechanism (EOIP), and the parallelism related mechanism Chunk-Level Parallelism (CLP). The experimental results show that these mechanisms can achieve considerable improvement on web retrieval latency.

Chapter 1 Introduction

1.1 Background and Motivations

1.1.1 Background

The *World Wide Web* (*web*) is the most popular application of the Internet [1]. The scale of the web has been experiencing exponential growth. Nowadays, the Internet traffic is dominated by web data transfers [2, 3, 4]. The web provides the most convenient way to distribute and access all sorts of information. Not only more and more companies and organizations turn to utilize the web to do their businesses, but a tremendous amount of users are also attracted to the web for their personal activities such as shopping, education, and entertainment etc.

With the explosive growth of the web, web retrieval latency has become one of the principal concerns to most web users and web content providers. Due to the immense amount of web traffic, the problems of congested network and heavy-loaded web servers become more and more serious. This results in long web retrieval latency, and thus the World Wide Web has been bantered as World Wide Wait. There is a commonly recognized “eight-second rule”, which indicates that after eight seconds of wait time, two thirds of the users of a website will be lost [5]. This rule is for 56k modem users. For broadband users, the tolerance level could be much lower. With the widespread commercialization of the web, exceeding the “eight-second rule” for downloading times would mean a significant loss in revenue. The businesses of web content providers depend on the ability to deliver information quickly to end users not only because speedy delivery will attract more users, but a faster content delivery also allows for more complex content which can provide a more enjoyable user experience. Therefore, faster and more efficiently means to access the web are preferred by both web users and web content providers.

Researchers have been working on how to improve web retrieval performance since the early 90's [6, 7]. There are basically two approaches to the acceleration of web retrieval. The first one is hardware approach which tries to accelerate web retrieval by improving the hardware capability of network infrastructure and bandwidth and the computing power of server and client machines. However, this approach has the following shortcomings which make it insufficient in solving the problem:

- The procedure of upgrading hardware infrastructure is usually very slow. For example, despite the great effort in improving network capacity, broad-band is still far from the Internet society. Nowadays, a significant percentage of web users still connect to the Internet through slow dial-up accounts.
- Upgrading of hardware infrastructure is not cost-effective. Improving hardware capability often means the purchase of pricey equipments, and it often can not solve the problem effectively. For example, upgrading a dial-up link to T1 or T3 lines may not completely solve the speed problem as the effective rates of the connections can be as slow as, or even slower than a dial-up connection when the T1 or T3 lines are shared by a lot of users.
- The requirement and expectation on web access grows much faster than the development of hardware. On one hand, websites have become bloated as content providers attempt to provide clients with more information. On the other hand, web users continue to expect more and more performance from their existing web links. A research indicates that although the Internet backbone capacity increases as high as 60% per year, the demand for bandwidth is still likely to outstrip supply in the foreseeable future [8].

If some other kinds of solutions are not undertaken for the problems caused by its

rapidly increasing growth, the web would become too congested and its entire appeal would eventually be lost. What comes into help is the second approach, i.e. the software approach. This approach is often referred to as web acceleration. It has little to do with the hardware. Web acceleration tries to integrate various software technologies and methodologies to get content from an origin server to an edge client as quickly as possible. Typical examples of web acceleration include web caching, prefetching, content optimization, and content delivery networks (CDN) etc. [9, 10, 11, 12, 13, 14, 15, 16].

With the maturity of techniques on web intermediate servers such as web proxies, web intermediaries are actively involved in web acceleration. Many researchers are looking into acceleration mechanisms that work on web intermediate servers. This direction has shown great potential because of its good cost-effectiveness, scalability and functionality.

Web content acceleration is an important method used to address the surge in web access, and it is believed to have better potential than hardware approach because not only it is more cost-effective, but it can also cater the needs of users from various environments. In this thesis, we focus our study on the issues of web acceleration.

1.1.2 Motivations

Web retrieval latency has been extensively studied and many acceleration mechanisms have been proposed. The most popular mechanisms are those caching-based schemes such as caching [9, 10, 11] and prefetching [12, 13, 14]. However, the performance of such acceleration mechanisms is limited due to the low reuse rate and poor cacheability of web objects [13, 17, 18, 19, 14, 20]. To overcome the limitation, researchers are actively looking into mechanisms which accelerate the downloading process of web retrieval. Examples of such mechanisms include

persistent connection [21, 22], bundling [23, 24, 25], and content transformation [26, 27, 28] etc.

Although many research works have shown good potential in web acceleration, they still have some deficiencies which motivate us to further look into this area. In detail, the motivations for the research work reported in this thesis come from the following deficiencies in the current studies:

- Lack of a precise model to capture web retrieval process precisely
- Lack of study at detailed levels of web data retrieval
- Lack of in-depth understanding and studying of page retrieval latency
- Lack of effective acceleration mechanisms with special emphasis on page retrieval latency

The current web content is made up of pages which usually consist of multiple web objects such as html, image and other types of files [29]. The basic unit of web browsing is web page. Therefore, page retrieval latency is more meaningful to web users than object retrieval latency. However, most previous works based on object retrieval latency to study web retrieval latency [30, 31, 32, 33, 34]. This is insufficient and sometimes inaccurate since the unit of web browsing is web page instead of object. While page retrieval latency is derived from object retrieval latency, the relationship between them is not that direct and simple. When objects are put together to form pages, more complex and interacted factors will be involved in determining the final page latency. Normally, in a web page, there is a primary object called container object, which contains the definitions of other objects (embedded objects) of the page. Because of this, the retrieval of the embedded objects highly depends on the retrieval process of the container object of the page, and this dependency will introduce significant delay to the retrieval process of the embedded objects. Furthermore, current

web system employs parallelism for parallel fetching of objects, which makes it possible for the retrieval of some objects to virtually have no effect on the total page latency. All these factors make the mapping from object latency to page latency very complicated, and they are largely ignored in previous object-level studies in web content delivery.

On the other hand, the transfer of web data is typically delivered in a sequence of data chunks. The characteristics of chunk sequence transfer have great impact on web retrieval latency. A thorough study on the detailed chunk level transfer would be very useful in helping user to better understand the root causes of web retrieval latency. However, such studies are rarely seen in existing research works.

To well understand and study the complex factors affecting web retrieval latency, especially page retrieval latency, we will need a more precise model. In this thesis, we address these issues by proposing a detailed operation level and chunk level model to provide precise capture of web retrieval process. Based on the model, we conduct comprehensive, in-depth studies on both detailed levels of web data transfer and whole page retrieval latency. We also propose new web acceleration mechanisms to improve web retrieval performance, especially whole page retrieval latency.

1.2 Thesis Aims

The focus of this thesis is to address some issues in web acceleration. Due to the performance limitation of caching-based mechanisms, we do not make it the heart of our study. Instead, we spend much of our effort on the studies which aim to accelerate the downloading process of web retrieval, with specific emphasis on whole page retrieval latency. The detailed aims of this thesis are originated from the motivations stated in the previous section, and they are described as follows.

Firstly, we propose a fine grained model to address the issue of lack of precise

model for studies in web retrieval. The model shall provide precise capture of web retrieval process at very detailed level so that it can be used for better understanding and study of web retrieval.

Next, we acquire better understanding of web retrieval latency for both objects and pages based on the model proposed. We expect to reveal the impact of detailed level operations and chunk transfers on object retrieval latency and the complex factors determining page retrieval latency. We also want to further demonstrate the deficiency of previous object-level studies by analyzing existing acceleration mechanisms. We would also like to derive upper bounds on the performance improvement for acceleration mechanisms to help us to understand the potentials of web acceleration.

Lastly, we propose new acceleration mechanisms with specific emphasis on improving page retrieval latency. The new acceleration mechanisms are originated from the findings from the studies based on our model, and we conduct comprehensive experiments to study the effectiveness of them.

1.3 Thesis Organization

The overall structure of this thesis is shown in Figure 1.1. After the introduction in Chapter 1, Chapter 2 reviews the related work in the web acceleration area; both research work and real acceleration systems are discussed. As web caching based mechanisms are still the important solutions to web acceleration, we include a study on it in this thesis, and it is presented in Chapter 3. We dig into the relationship among the co-occurrent factors to reveal the effectiveness of them in the co-occurrence situation, and investigate the accuracy of the settings for the TTLs of objects to reveal its impact on web caching.

Move on to the main part of the thesis, we first propose a fine grained Web Retrieval Dependency Model (WRDM) in Chapter 4, and conduct detailed study and

analysis on web retrieval latency based on this model in Chapter 6. Chapter 5 describes the tools, traces, environments and methodologies used for the studies in this thesis.

To further demonstrate the usefulness and effectiveness of our WRDM model, we analyze an important acceleration mechanism, namely web compression, in Chapter 7. The results reveal some important effect and implication of compression on page retrieval latency. Also in this chapter, we propose a new compression mechanism named content-aware global static compression to improve the performance of compression in web content delivery.

Based on the studies using our WRDM model, we propose some new mechanisms for web acceleration. Besides the novel compression mechanism proposed in the later part of Chapter 7, we also proposed and studied innovative acceleration mechanisms related to dependencies and parallelism in web retrieval in Chapter 8 and Chapter 9, respectively. Detailed descriptions and results are reported in these chapters.

Finally, the thesis concludes in Chapter 10. It briefly summarizes the work presented in the thesis and lists the main contributions of my work. Some future works for making further contributions to this area are also discussed in this final chapter.

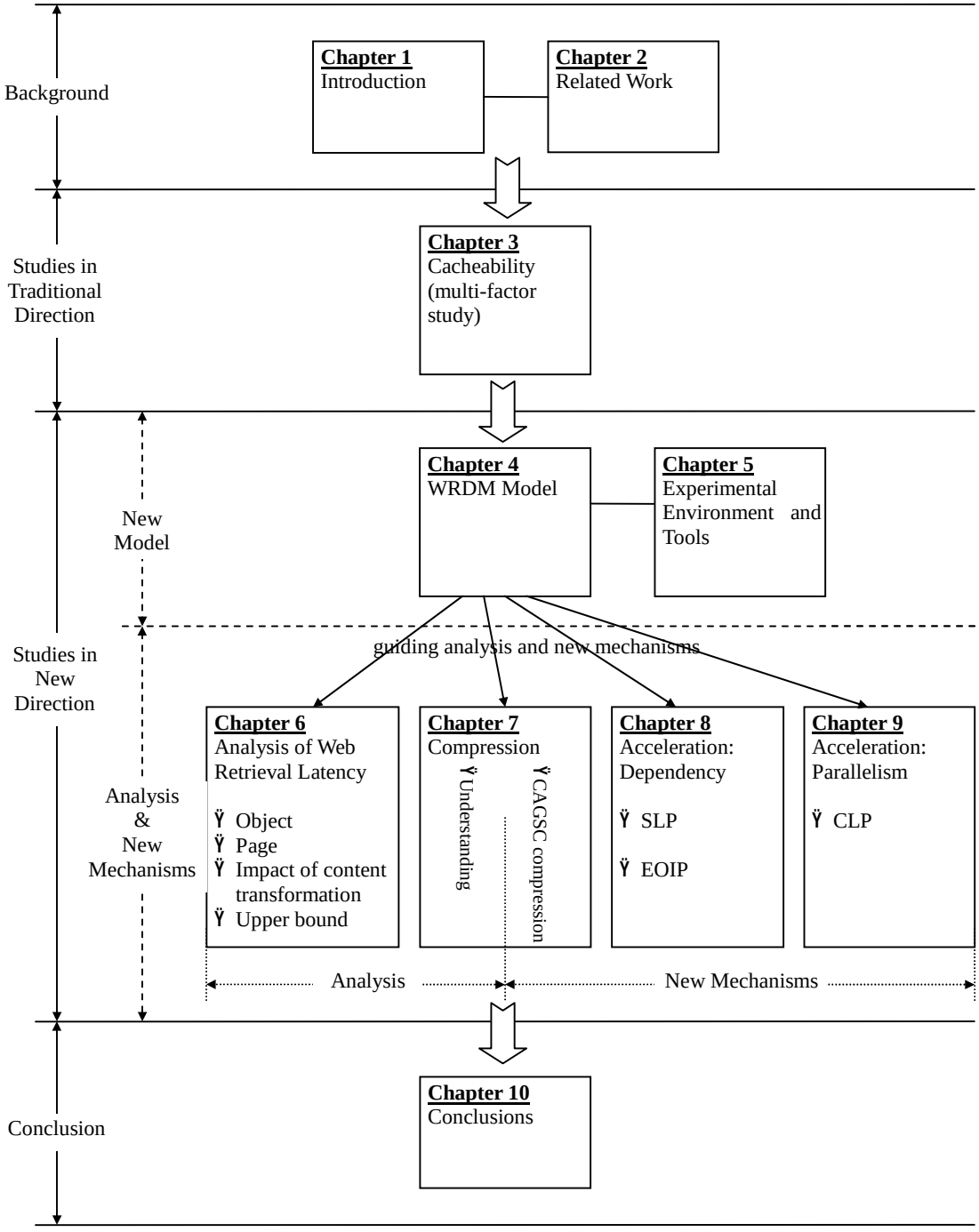


Figure 1.1 Structure of the thesis

Below are the papers I have finished during my study. The papers cover my research work from processor cache system to web caching system, and then non-caching based web acceleration studies. I was the main contributor for most of the papers, especially those published since 2002.

- Multi-factor Effect of Cacheability Factors (with Chi-Hung Chi), (Submitted)
- Content-Aware Global Static Compression for Web Content Delivery (with Chi-Hung Chi), The IEEE Tenth International Workshop on Web Content Caching and Distribution (WCW 2005), Sophia Antipolis, French Riviera, France, September 12-13, 2005.
- Exploiting Fine Grained Parallelism for Acceleration of Web Retrieval (with Chi-Hung Chi and Qibin Sun), The Third International Human.Society@Internet Conference (HSI'05), Tokyo, Japan, July 27-29, 2005. (The conference proceeding was published by Springer Verlag in Lecture Notes in Computer Science series, July 2005.)
- A More Precise Model for Web Retrieval (with Chi-Hung Chi and Qibin Sun), The Fourteenth International World Wide Web Conference (WWW 2005), Chiba, Japan, 10-14 May 2005.
- Understanding the Impact of Compression on Web Retrieval Performance (with Xiang Li and Chi-Hung Chi), The Eleventh Australasian World Wide Web Conference (AusWeb'05), Gold Coast, Queensland, Australia, 2-6 July 2005.
- Modeling Retrieval Parallelism in Web Content Delivery (with Chi-Hung Chi and Qibin Sun), The 2005 International Symposium on Web Services and Applications (ISWS'05), Las Vegas, Nevada, USA, June 27-30, 2005.
- Unveiling the Performance Impact of Lossless Compression to Web Page Content Delivery (with Chi-Hung Chi), The Ninth International Workshop on Web Content

Caching and Distribution (WCW 2004), Beijing, China, 18-20 October 2004. (The conference proceeding was published by Springer Verlag in Lecture Notes in Computer Science series, Volume 3293/2004.)

- Web Caching Performance: How Much Is Lost Unwarily? (with Chi-Hung Chi), The Second International Human.Society@Internet Conference (HSI'03), Seoul, Korea, June 18 - 20. (The conference proceeding was published by Springer Verlag in Lecture Notes in Computer Science series, Volume 2713/2003.)
- Runtime Association of Software Prefetch Control to Memory Access Instructions (with Chi-Hung Chi), The Eighth International Euro-Par Conference (Euro-Par 2002), Paderborn, Germany, August 27-30, 2002. (The conference proceeding was published by Springer Verlag in Lecture Notes in Computer Science series, Volume 2400/2002.)
- Load-balancing Data Prefetching Techniques (with Chi-Hung Chi), Journal of Future Generation Computer Systems (FGCS), 17(6):733-744, 2001. (Invited paper)
- Load-Balancing Branch Target Cache and Prefetch Buffer (with Chi-Hung Chi), The 1999 IEEE International Conference on Computer Design (ICCD 1999), Austin, Texas, USA, October 10-13, 1999.
- Sequential Unification and Aggressive Lookahead Mechanisms for Data Memory Accesses (with Chi-Hung Chi), The Fifth International Conference on Parallel Computing Technologies (PaCT-99), St. Petersburg, Russia, September 6-10, 1999. (The Conference Proceedings were published by Springer Verlag in Lecture Notes in Computer Science series, Volume 1662/1999.)
- Design Considerations of High Performance Data Cache with Prefetching (with Chi-Hung Chi), The Fifth International Euro-Par Conference (Euro-Par 1999), Toulouse, France, 31 August - 3 September 1999. (The conference proceeding was

published by Springer Verlag in Lecture Notes in Computer Science series, Volume 1685/1999.)

- Cyclic Dependence Based Data Reference Prediction (with Chi-Hung Chi and Chin-Ming Cheung), The Thirteenth International Conference on Supercomputing, Rhodes, Greece, June 20-25, 1999.

Chapter 2 Related Work

2.1 Introduction

The *World Wide Web* (*web*) was initially introduced to the public in 1991 [6, 7]. The web system is built on a number of protocols and languages. Among them, the most important ones are the *HyperText Markup Language* (*HTML*) and the *HyperText Transfer Protocol* (*HTTP*) [35, 36, 37]. HTML is the basic tool to specify the semantics and structure of web information. It is commonly used to describe the content and presentation of web objects and pages. HTML files are in simple textual format. The most popular version of HTML is 4.0 series in current web system. The HTTP protocol is layered over a reliable bidirectional byte stream, normally TCP [38]. Each HTTP interaction consists of a request sent from the client to the server, followed by a response sent from the server to the client. Requests and responses are expressed in a simple ASCII format. There are mainly two versions of HTTP in current web system, i.e. HTTP/1.0 and HTTP/1.1. While the 1.1 version is getting its popularity, the 1.0 version of HTTP is still used widely in current web system.

With the evolution of the web, there emerge a number of new languages and protocols. Typical languages are represented by the Extensible Markup Language (XML) [39, 40], Wireless Markup Language (WML) [41, 42], Edge Side Includes (ESI) [43, 44], and Web Service Description Language (WSDL) [45] etc. Protocols examples include the Internet Cache Protocol (ICP) [46, 47], the Hyper Text Caching Protocol (HTCP) [48], the Internet Content Adaptation Protocol (I-CAP) [49, 50], the Open Pluggable Edge Services (OPES) [51, 52, 53, 54], the Simple Object Access Protocol (SOAP) [55, 56], Web Intermediaries (WEBI) [57, 58, 59], Web Replication and Caching (WREC) [60, 61], Middlebox Communication (MIDCOM) [62, 63], and Reliable Server Pooling (RSERPOOL) [64, 65, 66, 67, 68, 69] etc.

All of the new languages and protocols aim to improve the application or performance of the web in one way or another. But up to now, the majority of them still have not got their popularity yet. The web traffic nowadays is still dominated by HTTP and HTML. So, in this thesis, we will focus our study on HTTP and HTML. However, most of our works will be applicable to other languages and protocols as well.

Web content is usually made up of various types of objects such as html, image and other types of files. Many of the web objects exist before they are requested. Such objects are referred to as static objects. In recent years, another type of objects, namely dynamic objects become prevalent. Dynamic objects mainly refer to those objects which are generated in real-time when they are requested. Typical examples of dynamic objects include those generated by cgi, asp, or jsp programs.

While web object is the basic unit of web content, it is not the basic unit of web browsing. In current web system, the basic unit of web browsing is web page. A web page is often made of multiple objects. Among the objects in a page, there is one primary object corresponding to the URL (Uniform Resource Locator) of the page. This object is called *Container Object (CO)* and is generally described in HTML language. The other objects in the page are called *Embedded Objects (EO)* which have their definitions (usually URLs) found in the body of the container object. When a web page is requested, the CO of the page will be first returned to client. Then, client will see the definitions of the EOs and subsequently send requests from them. The content of both the CO and EOs are interpreted and displayed together to render the full view of the web page.

The web system is running in a client-server model. There are numerous web servers and clients connected in the Internet. Clients run web browsers like MS-IE and

Netscape [70, 71] which initiate web retrieval by sending requests to web servers. Web servers are typically represented by Apache, MS-IIS and Netscape web server etc. [72, 70, 71]. They manage web content and process requests from clients. On receiving a request from a client, the server will find or generate the content corresponding to the request and send it back to the client.

Besides the servers and clients, there are also intermediate servers widely deployed in the web system. These intermediate servers are commonly known as proxy servers or middle-boxes. They are introduced to improve various issues of web system, such as performance, security, and scalability etc. Examples of such intermediaries include Squid [73, 74] and W3C httpd [75] etc.

All web retrievals undergo certain latencies. Some of the latency comes from the physical limitation of the machines and network such as the computing power, the network bandwidth and the propagation speed limit of electronic signal. Some other parts of the latency come from the operations and mechanisms of the retrieval process such as the establishment of network connection and the parallelism in web retrieval etc.

As the web continues its exponential growth, the problems of congested network traffic and long web retrieval latency become one of the principal concerns to most web users and web content providers. Hence, the acceleration of web retrieval has become a primary focus of the Internet research and development community.

The studies on web acceleration in the literature are extensive. Most early studies focused on web caching and prefetching related area such as cache replacement algorithms [76, 17, 30, 77], cacheability of objects [78, 79, 80, 20], cache consistency issues [81, 82, 83], and prefetching algorithms [84, 85, 86, 87, 88, 89] etc. The mechanisms in this direction are actually based on caching to accelerate web retrieval.

However, recent studies show that the performance of such mechanisms is limited because of the low reuse rate and poor cacheability of web objects [13, 14, 17, 18, 19, 20]. To overcome the limitation, researchers are actively looking into a new direction which tries to accelerate the downloading process of web retrieval. Example mechanisms in this direction include persistent connection [22, 37], bundling [23, 24, 25], content transformation [26, 27, 28] etc. The studies in this direction have shown promising potential of improvement in web retrieval latency. However, most of them only focus on object latency. As page is the basic unit of web browsing, it would be more important and meaningful to study page latency instead of just object latency. Nevertheless, the modeling and acceleration of page retrieval is still a missing link in current studies.

As the application and population of the web grow explosively, the traffic on the web grows much faster than the growth of underlying network hardware and machine's computing power. Moreover, the growth of users' expectation on the performance of web retrieval seems to always outstrip the growth of the Internet backbone capacity. All these make the need of web acceleration become even more urgent. What is more, with the growth of mobile devices and wireless networking, the demand for good performance for pervasive Internet access arises. This gives even tougher challenges to web content delivery as the computing power and bandwidth in these environments are quite different from the traditional web system. Thus, great efforts are still needed to solve the problems.

In this chapter, we would like to review the related work in the area of web acceleration.

2.2 Related Work in Caching-based Acceleration Mechanisms

2.2.1 Basics of Caching

Web caching is the first major technique that attempted to improve performance, reduce latency, and save network bandwidth. However, the idea of caching is nothing new. It originates from the long-standing use of caching in memory architectures, where this principle is used to speed up memory access by storing data in a small amount of high speed memory close to CPU [90, 91, 92, 93]. Due to the two locality characteristics of requests, i.e. temporal locality and spatial locality, the data brought into the cache by previous requests can often be used to serve future requests. The "caching" in the context of web system performs similar function. It tries to improve the performance of web retrieval by storing copies of objects in local storage and using them to serve future requests. Because the objects are served locally, so the retrieval latency can be reduced and external network traffic can be saved.

Web caching can be used in a number of places throughout the web system. First of all, web browsers may implement their own caches on disk and/or in memory. However, the performance of such web caches is not good because of the low reuse rate of web objects since such caches are used by single or few users. A better place for web cache is a network point shared by multiple users. This is typically the gateway point or the ISP of an organization. The web caching function performed here is often incorporated with proxy function, and together they are called web proxy server. The web caching in the proxy server can produce better performance because it serves multiple users so that the reuse rate of web objects could be much higher than those for single users. In some cases, web caching function is also performed right in front of web servers to improve the performance of them. Again, it is also often combined with proxy function. Such proxy servers are often referred to as *reverse proxy servers*. In

contrast, those proxy servers close to end users within an organization are referred to as *forward proxy servers*.

Web caching has become a significant part of the infrastructure of the web. It even led to the creation of a new industry: *Content Delivery Networks* (*CDNs*). CDNs rely on web caching and load-balancing technologies to efficiently deliver large amounts of data over the web. The market value of CDN grows at a fantastic rate, which expects to be over three billion US dollars in sales and services by 2006 [94]. This reflects the importance of caching in the web system.

Ordinary caching reduces latency only for repeated requests. Prefetching is a supplementary technique to caching. It aims to predict future user requests and prefetch the objects into the cache in advance so that more requests, including those first time requests and repeated requests, can be satisfied. The concept of prefetching is not new either. Many advanced computer systems use this principle to improve the performance of the memory architectures [95, 96, 97, 98, 99, 100, 101]. Although the idea is similar, the prefetching in the context of web system is more difficult than that in computer memory system. The challenge lies in that the user requests are not so predictable as the memory accesses in computer memory system. It is difficult to achieve high prediction accuracy in web prefetching.

There are many issues in the web caching area, and they have been extensively studied in the current literature. Below, we examine the major works in this area.

2.2.2 Locality of Web Requests and Cacheability of Web Objects

The locality of web requests reflects the reuse rate of objects, and the cacheability of web objects refers to the availability and duration that web objects can be kept in a web cache. These two factors are very fundamental to caching-based acceleration mechanisms because caching is only effective when there is fair reuse rate

and good cacheability of objects.

Cao [102], Breslau [103] and Dykes [104] et al studied the Zipf-like distribution of web requests, which states that the request frequency for a web object is inversely proportional to the object's popularity ranking. Abdulla et al pointed out that web traffic has a significant daily and weekly cyclic component, and claimed that the temporal and spatial locality of reference within examined user communities is high, so caching can be an effective tool [105, 106, 107]. Cao and Irani [30] found a large number of repeat requests in their studies. [18, 13, 9, 108] and [109] etc reported fair object reuse rate, ranging from 24% to 45%. [110] further pointed out that embedded images in web pages are often reused, even the pages change frequently. Zhang [111] found that between 15% and 40% of web objects in their traces can not be cached, and Dykes, Robbins, and Jeffery [78, 79, 80] reported that 28% of the successful GET requests are for non-cacheable documents. Many of the caching mechanism in the web depend on HTTP header fields that carry absolute timestamp values to determine the cacheability of objects. Wills [112] and Mogul [113] examined the effect of those timestamp-based cacheability-controlling HTTP headers and showed that many objects are not cacheable due to inaccurate and nonexistent directives. If such errors can be corrected, more objects will be turned to be cacheable.

2.2.3 Cache Replacement Algorithms

Cache replacement algorithms govern the eviction of old objects from the cache when there is not enough space to store new objects. Different replacement algorithms may yield different hit rates and byte hit rates. So, replacement algorithm is one of the key aspects that ensure the effectiveness of web caching.

The traditional replacement algorithms like Least Recently Used (LRU) and Least Frequently Used (LFU) widely used in computer memory architectures are also

imported into web caching systems. Williams et al gave an extended algorithm based on LRU: Pitkow/Recker [76]. In this algorithm, objects are evicted in LRU order except for those objects accessed within the same day, where the largest object will be evicted. The rationale behind this algorithm is that they found that a caching algorithm based upon the recency rates of prior document access could reliably handle future document requests.

Some other replacement algorithms specially developed for web caching are based on some key properties of objects such as size. The algorithm SIZE evicts the largest objects [76]. LRU-MIN and LRU-Threshold have a certain threshold size to guide the eviction of objects [17].

Another category of replacement algorithms for web caching typically takes into consideration the timing or latency factors. A cost function is derived from those factors to govern the eviction of objects. Cao et al proposed the GreedyDual-Size (GDS) algorithm [30, 77]. It associates a cost with each object and evicts object with the lowest cost/size ratio. Because it incorporates the latency and size concerns, this algorithm yields better performance in terms of latency reduction and network cost reduction. There is a number of works trying to further improve the performance of GreedyDual-Size algorithm. Cherkasova proposed the Greedy-Dual-Size-Frequency (GDSF) and the Greedy-Dual-Frequency (GDF) algorithms, which incorporated different characterizations of objects such as size, access frequency and recentness etc [114]. Jin and Bestavros first proposed the Popularity-Aware GreedyDual-Size algorithm [115], which makes use of popularity profile of web objects. They later proposed the GreedyDual* algorithm, which is said to be a generalization of GreedyDual-Size [116]. The GreedyDual* algorithm capitalizes on and adapts to the relative strengths of both long-term popularity and short-term temporal correlation.

There are still a number of other replacement algorithms such as Hyper-G [76], Lowest Latency First [32], Hybrid [32], Lowest Relative Value (LRV) [117, 118], LNC-W3 [119] etc. However, the performance of replacement algorithms depends highly on traffic characteristics of web accesses. No known algorithm can outperform others for all web accesses patterns. Therefore, many current web caching systems still widely use the traditional replacement algorithms like LRU [120].

2.2.4 Cache Coherence and Validation of Objects

Cache coherence is concerned with ensuring that the cached objects do not reflect stale or defunct data. Web cache relies on some timestamp-based HTTP headers like Data, Last-Modified and Expires etc. to determine the freshness of objects [121]. There must be some mechanisms to assure the validity of cached objects when their master copies on the web servers change. This is typically the validation/invalidation process in web caching systems.

The validation process is normally initiated by web caches. A web cache sends an If-Modified-Since message to the server to verify the validity of an object. The server either returns a “Not-Modified” message to assure the validity, or returns a new copy of the object if it has been changed [37, 81, 82]. This process can be performed either for each access, or periodically only when an object is suspected to be stale [83]. The latter improves access latency, but may not be able to maintain strong coherence.

Instead of having web caches to check for the validity, web servers can also send invalidation messages to all clients upon detecting changes of objects [121]. This approach requires a server to keep track of the web caches that are caching its objects and contact them when objects change. When the number of web caches contacting a server is big, this task can become unmanageable for the server.

A number of works also have been done to improve the effectiveness of

validation and invalidation processes. First, the Adaptive TTL policy is proposed to adjust the time-to-live of objects and it is shown to be able to keep the probability of stale objects within reasonable bounds (< 5%) [122, 123, 124]. Another direction is to piggyback the validation or invalidation message to an existing communication between the server and cache. The ideas, Piggyback Cache Validation (PCV) and Piggyback Server Invalidation (PSI), are explored by Krishnamurthy and Wills [125, 126]. Their studies show that PCV and PSI minimize access latency and bandwidth usage while maintain a close-to-strong coherence. Mikhailov and Wills also proposed an alternative approach to strong cache consistency called MONARCH. They showed that MONARCH does not require servers to maintain per-client state and it generates little more request traffic than an optimal cache coherency policy [127].

2.2.5 Prefetching

The performance of ordinary web caching is limited due to the relatively low reuse rate of objects, typically ranging from 24% to 45% as reported in many studies [18, 13, 9, 108, 109]. Prefetching is an important method to help further increase cache hit ratio. By predicting future user requests and prefetch the objects into the cache in advance, it can satisfy more user requests.

The prefetching in the context of web system has a significant difficulty which is the accuracy of prediction. Because web users' requests are not so predictable as the memory accesses in computer memory system, it is often difficult to achieve high prediction accuracy in web prefetching. While a few works try to prefetch only inline objects of pages where the accuracy is not an issue [128], most other studies on web prefetching focus on improving the accuracy of prediction algorithms.

A naïve method for doing web prefetching is to have proxy cache to fetch all the pages that are pointed to by the hyperlinks in current page. So, no matter which page

the user goes next, there will be always a cache hit. [129] proposed a prefetching scheme of this type and reported significant improvement in cache hit ratio. However, this method imposes too heavy load on the web system.

More advanced prefetching algorithms employ the knowledge of data mining or mathematics to do the prediction, and the information used for prediction may come from client side or server side. Maltzahn et al applied machine learning techniques to automatically develop prefetch strategies and showed that the results are promising [84, 85]. Palpanas and Mendelzon investigated the use of partial match prediction, a technique taken from the data compression literature, for prefetching in the web [86]. Their results suggest that a high fraction of the predictions are accurate, e.g., predicts 18-23% of the requests with 80-90% accuracy.

Sarukkai sought to apply Markov chains to predicting web requests and claimed a lot of promise [87]. Deshpande and Karypis also studied Markov models for predicting web page accesses [88]. They studied different techniques for intelligently selecting parts of different order Markov models to reduce the state complexity of the model and to improve prediction accuracy. Their results indicate that the performance of their model is consistently superior to that obtained by higher-order Markov models.

Markatos and Chronaki proposed a top 10 approach to prefetching which combines the servers' active knowledge of their most popular documents (their top 10) with client access profiles [89]. According to these profiles, clients request and servers forward to them their most popular documents. Their results suggest that the top 10 approach can anticipate more than 40% of a client’s requests while increasing network traffic by no more than 10% in most cases.

There are numerous other studies trying to work on web prefetching algorithms [14, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143]. Also, there

are some works studied the effects of prefetching on network and server, and the potential and limits of prefetching [144, 145, 146, 147, 148]. Among the studies, Davison argued that the current support for prefetching in HTTP/1.1 is insufficient because prefetching with GET is not good [145]. Pandey et al conducted a comparative study of some prefetching models and found that the model based on higher order page interaction is more robust and gives competitive performance in a variety of situations [149].

2.2.6 Others Aspects of Caching

There are many studies addressing other miscellaneous aspects of web caching. Cache hierarchy related issues and standards are discussed in [46, 47, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160] etc.

Kelly, Mogul, Bahn, Lee et al studied the aliasing/replica problem that affects the performance of web caching [161, 162, 163, 164]. Their studies revealed a significant percentage of web objects encounter the aliasing problem, which considerably lower the performance of web caching systems because they generally treat the replicas as different objects since they have different URLs. Different schemes are proposed to remove the redundant objects from web cache so as to improve caching performance.

As dynamic and secured data become more and more popular in the web, researchers are actively looking into way to cache such data. Many dynamic caching and active caching mechanisms have been proposed in an attempt to address this issue [165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177]. The studies indicate a lot of promise in this direction.

Content Distribution Networks (CDNs) are managed networks of caching and networking systems, and rely on web caching (as well as load-balancing) to efficiently deliver large amounts of data over the web. There are also a number of works trying to

address the issues with CDNs such as consistency management, request redirection, and object replication strategies etc. [178, 179, 180, 181, 182, 183, 184, 185]. With these issues being addressed properly, the performance of CDNs will reach even a higher level.

An important movement about CDN is that the technology has been exploited for more than web caching and multimedia content delivery recently. The largest traditional application for CDNs is the acceleration of web content delivery, along with streaming audio and video. But as more and more companies move their corporate applications onto the web, the CDN technology has been exploited for distributed computing and application acceleration, which is being shown to be a much bigger opportunity for CDNs than just helping on web site access [15, 16].

There are also some studies examining the effect of special activities on web caching, such as web robots, connection aborts and cookies etc [186, 187]. Results show that such details can affect the performance of web caching considerably.

2.3 Related Work in Other Acceleration Mechanisms

The performance of web caching is limited due to the low reuse rate and poor cacheability of web objects [9, 18, 13, 108, 109]. The ever-growing dynamic content in the web system further worsens this situation. To overcome the limitation, researchers are actively looking into a new direction which tries to accelerate the downloading process of web retrieval. This direction has huge potentials because it covers all pages and objects, independent of objects' reuse rate and cacheability [188, 189]. The mechanisms from this direction mainly fall into two categories: the first category aims to reduce network connectivity related latency; the other one endeavors to reduce the latency come from the actual transfer process. In this section, we review the major works in this direction.

2.3.1 Connectivity Related Mechanisms

Web retrieval usually involves some network connectivity related operations such as DNS resolution and connection establishment. Many studies revealed that such connectivity related operations contribute a significant portion to web retrieval latency [24, 33, 190, 191, 192, 193].

The contribution of DNS lookup costs to web retrieval is studied and addressed by [193, 194, 195, 33, 196] etc. Typical mechanisms proposed include stored address binding [194], pre-performing DNS lookups [195], proactive caching of DNS records [196] and so on. Considerable improvement is reported from this aspect.

To address inefficiencies associated with multiple concurrent connections, persistent-connection is first proposed [22, 24, 37, 197, 198, 192, 199, 200]. By keeping a connection alive and reusing it for pipelining a set of requests and objects transfers, persistent-connection can greatly reduce the response time, server overheads, and network overheads of web retrieval. Connection caching is further proposed to handle the connection management problem [192, 198, 200].

Pre-connection is another way to directly address the connectivity issue. This mechanism tries to pre-setup connections prior to the issuance of HTTP request. Results showed moderate (about 6%) performance improvement with pre-connection [195].

Another quite different approach to addressing the connectivity problem is bundling techniques such as GETLIST [24], MGET [23] and N-to-1 Bundle [22, 25, 201, 202]. By packaging a set of associated embedded objects into a single bundle for retrieval, bundling reduces the need for multiple requests and the load on the network. Thus network connections can be utilized more efficiently and web servers can have better control over the number and duration of connections they support.

2.3.2 Transfer Related Mechanisms

The mechanisms in this category aim to reduce the latency come from the actual transfer process. The basic idea is to reduce the amount of data being transferred in web retrieval. Examples include encoding, transcoding and content optimization etc. Some of the techniques mainly apply on textual objects, while some others are mainly for image objects.

The most popular encoding mechanisms include delta encoding and compression. The ESI (Edge Side Includes) can also be regarded as a special kind of encoding. Delta encoding and ESI enable web caches to retrieve only the difference (or delta) between the old instance and the new instance of an object instead of the entire new instance, and apply the delta to the cached copy of the old instance to construct the new version. Studies show that the reduction in response size and delay is significant [26, 27, 28, 44, 203, 204, 205].

Compression mainly refers to applying lossless compression algorithm on textual web objects. The support for such compression has been included in protocols and web servers and browsers [36, 37, 206, 207, 208]. In current web system, the supported compression applications/algorithms are gzip, compress, and deflate etc. [37, 209], which are mainly LZW or LZ77 based algorithms [209, 210, 211, 212, 213].

Nielsen et al reports on the benefits of compression in HTTP [191]. They observed over 60% gains in downloading time in low-bandwidth environment by using the zlib compression library [214] to pre-compress HTML files. Mogul et al [27] studied the potential benefits of delta encoding and data compression for HTTP. They reported about 35% reduction in transferred size and about 20% reduction in retrieval time when gzip compression is used. They also studied modem-based compression and pointed out that high-level compression seems almost always performing better than

modem compression. Other studies reported that compression could achieve up to above 90% reduction in file size and above 60% reduction in downloading time [203, 206, 215, 216, 217].

Transcoding mainly applies on image objects, often by using lossy-compression on images. The challenge lies in finding good compression algorithm which can make file sizes as small as possible while retaining the visual integrity of the images. The effect of transcoding on web retrieval has been studied in a number of works and good results were reported [218, 219, 220, 218, 221, 219, 222, 223].

Content optimization typically performs on HTML files. When HTML files are created (mostly by dedicated editors), they are usually not optimized for transfer over the web because they often contain non-renderable data which takes much space. The advent of active server pages and XML etc has increased the web page size even more enormously. Examples of such unnecessary non-renderable data include HTML comments, notes from publishing tools, superfluous tags, carriage returns, and extra white spaces etc, and they can account for as much as 15% of the information in web retrieval [224]. The simple way to optimize web content is to remove unnecessary data that are not needed to properly render the page. A more comprehensive way is content selection. In this mechanism, some parts of web information can be selected for processing and others can be suppressed according to different environments and requirements. Studies as well as industrial practice have shown promising results in this direction [225, 226, 227, 228].

2.3.3 Others Mechanisms

There are still some other works trying to reduce the actual transfer latency. Parallel fetching of multiple objects can be used to reduce both the connectivity related and transfer related latencies by hiding the latencies among the requests. Most

common web browsers like Microsoft IE and Netscape already employed this parallel fetching for retrieving the objects in the same page. Rodriguez et al proposed paraloading [229, 230, 231], which suggested parallel segmented download of an object using parallel connections to multiple mirrors. While the use of parallel fetching can reduce user-perceived retrieval latency in many cases, it incurs additional overheads and puts higher workload on the network and the servers.

Pushing is a counterpart mechanism of prefetching. In contrast to prefetching which is usually done by clients, pushing is initiated by servers. In pushing, servers select the objects and timings to send data to clients without additional client requests. Because servers have better knowledge of the access patterns to its objects and its workload, pushing can potentially perform better than prefetching. When performed properly, pushing speed-up latency as high as 3.35 compared to the normal retrieval process [194, 232].

There are also some more aggressive approaches to web acceleration, which include the development of new data formats and new protocols. An example of the new data formats is the progressive image format such as Progressive JPEG and JPEG 2000 [233, 234, 235, 236]. Progressive image formats improve user-perceived retrieval latency by allowing web users to see an approximated image in its whole without the need to wait for the complete set of the data to be received.

Current web system is built on the HTTP protocol which is over TCP [38]. There are many performance issues against this combination. Due to the significant setup costs of TCP connection, HTTP is known to be inefficient for transfers of small objects. Furthermore, TCP is strictly ordered in the way it delivers packets, which could introduce considerable delay when packet loss occurs. Also, many acceleration mechanisms like caching and prefetching are not well supported by current protocols.

A number of attempts have been made to develop better protocols for web retrieval. Examples include WebMUX, Internet Cache Protocol, Hyper Text Caching Protocol, and Hypertext Streaming Transport Protocol etc. [46, 47, 48, 237, 238, 239, 240, 241, 242].

Finally, some works are also done on peer-to-peer web system. This is a relatively new way to improve web performance. This direction tries to exploit peer-to-peer techniques to improve the availability, scalability, latency etc. for web service. Some studies already show improvements in latency and reliability in such systems [243, 244, 245].

2.4 Existing Web Acceleration Systems

In this section, we take a look at a number of existing web acceleration systems. The list here is not exhaustive and it serves only as a brief introduction of some typical systems used in the web society.

2.4.1 Caching and Prefetching Systems

The first caching system worth of mentioning is Squid [73, 74]. Squid is one of the most popular web caching softwares in use today. It is a free, open-source, full-featured and high-performance web proxy cache designed to run on a variety of platforms including Linux, FreeBSD, and Microsoft Windows. Squid improves network performance by reducing the amount of bandwidth used when surfing the web. It makes web pages load faster, and can also be used as reverse proxy server to reduce the load on web servers. By caching and reusing popular web content, Squid allows web users to get by with smaller network connections.

Unlike traditional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process. It keeps meta data and especially hot objects cached in RAM, caches DNS lookups, supports non-blocking DNS lookups, and implements

negative caching for failed requests. All these reduce the amount of time required for caching processing. Squid supports many protocols like HTTP, FTP, gopher, SSL, ICP, HTCP, CARP, WCCP, etc. It can be configured to support transparent caching. A group of Squid caches can also be arranged in a hierarchy or mesh for additional bandwidth savings.

Another existing caching system is ProxySG from the Blue Coat Systems, which is formerly known as CacheFlow [246]. Blue Coat's ProxySG proxy appliance is a powerful, full featured solution for both forward and reverse proxy applications. Its unique Web Knowledge Framework enables it to handle all web protocols, including HTTP, HTTPS, FTP, Microsoft streaming (MMS and HTTP streaming), Real streaming (RTSP and HTTP streaming), QuickTime streaming (over RTSP), MP3, Flash, and many other web object types. Unlike traditional proxy caches, Blue Coat Systems has incorporated enhanced security features in its products as the security issues in current web system become more and more important The ProxySG appliance integrates advanced proxy functionality with optional security services such as content filtering, instant messaging (IM) control, Peer to Peer (P2P) control, and web virus scanning. Its special Policy Process Engine provides the power to define a comprehensive set of rules for protection, control and acceleration. The ProxySG proxy cache can be configured for controlling and securing user communications and applications over the web. It can manage user requests for content delivery effectively.

The NetCache from Network Appliance is a versatile proxy cache system which accelerates web access while minimizing bandwidth needs [247]. It supports many protocols like HTTP and FTP, and streaming media content as well. NetCache offers unique cacheability controls to maximize business application delivery performance and bandwidth savings. Many applications include a significant amount of dynamic, or

non-cacheable, content, which could be cached without disrupting the application. NetCache's unique controls enable the user to fine-tune the way NetCache handles such dynamic content for further improvement in web retrieval performance.

Some other available academic and commercial caching systems include CERN Proxy, Cisco Cache Engine, Novell Internet Caching System, InfoLibria DynaCache, etc. They all provide web caching function which help to accelerate content delivery in the web [75, 248, 249, 250].

Unlike caching products which are quite abundant in the market, products with prefetching capability are rarely seen. This is mainly due to the reason that prefetching technology is not mature yet. Currently, there is no prefetching mechanism which is obviously superior to others in terms of accuracy of prediction and extra network traffic incurred. SkyCache once provided a prefetching system, in which a large scale of web caches at Internet Service Providers (ISPs) are maintained and the most popular and up-to-date web content is continually broadcast to them over satellite link [251]. The popularity information is collected from the access statistics of each ISP cache. By using dedicated satellite links for broadcast the information for prefetching, it avoids network congestion points and relieves traditional links from high bandwidth prefetch traffic. Some of CacheFlow's (now known as Blue Coat) products also implemented a conservative prefetching, where only cached popular web objects are checked and prefetched (i.e. updated) [252]. The product uses access history information to determine the popularity of objects, and then it checks the popular objects and prefetches new copies of them when they become stale.

2.4.2 Content Delivery Network Systems (CDNs)

Content Delivery Network Systems (CDNs) rely on web caching and load-balancing technologies to efficiently deliver large amounts of data over the web.

The most famous CDN service provider is Akamai [253]. Akamai initially garnered much success with its FreeFlow static content delivery service. But as dynamic content becomes more and more popular in the web, it now focuses more on serving enterprise customers' dynamic content delivery needs with its EdgeSuite service. The core of Akamai's technologies is distributed web architecture. By placing thousands of servers at the edge of the network and caching content close to the users, optimal web content delivery performance can be achieved. Today, Akamai's edge platform for content, streaming media, and application delivery comprises more than 13,500 servers in more than 1,000 networks in 66 countries. This ensures high performance, reliability and scalability of web content delivery.

SAVVIS Communications also provides content delivery network services, which is the second largest CDN in the market today according to InStat MDR [254, 255]. The SAVVIS CDN service features a full range of flexible, easy-to-implement services including content delivery, streaming media production and delivery, traffic management and global load balancing. It also has a unique feature which enables true end-to-end delivery of all the necessary applications related to the creation, management and distribution of digital content. SAVVIS' CDN services enable enterprises to deliver digital content assets to end-users rapidly, reliably and cost-effectively.

There are other companies also providing CDN services such as Digital Island (merged with Sandpiper Networks and later Cable & Wireless), Inktomi (bought by Yahoo) and Maven Networks etc [256]. A number of vendors like Cisco, F5 Networks, Nortel and Volera offer CDN products, which include caching devices, web switches, and appliances for directing and scheduling content distribution throughout a corporation [248, 257].

2.4.3 Other Acceleration Systems

The support for persistent connection has been built into most today's web systems. But because HTTP/1.0, which does not support persistent connection, is still widely used in current web system, the advantage of persistent connection has not been fully utilized yet.

Some products try to improve connectivity related latency by managing TCP/IP connections. NetScaler provides products with advanced TCP optimization feature [258]. Their TCP offload technology optimizes server-side resources to achieve higher throughput. By managing and accelerating the fundamental TCP/IP connections, NetScaler system can multiplex the basic TCP/IP connections, which leads to significant improvement in the efficiency of web retrieval. Redline Networks' also provides products which can multiplexing each set of requests onto one connection [259]. This enables them to take the advantage of the inherent architecture of TCP, which is designed to transfer data in larger bytes over one channel, rather than several little channels. Their acceleration tests showed that user access to both dynamic and static content can be boosted up to a factor of four.

Compression feature can be found in many web systems today. Most web browsers such as Microsoft IE and Netscape have also equipped support for web compression since 1998 [206]. Web servers like Microsoft IIS 5.0 [207] and Apache [208] have also incorporated compression capability. NetScaler's AppCompress provides real-time compression for both encrypted and unencrypted application data. This reduces last mile transmission times and dramatically improves user-perceived latency [258]. Products from Packeteer [215], Redlinenetworks [259] and BPVN Technologies [260] all have similar compression feature.

Transcoding is the technology used to accelerate web access especially for

pervasive Internet access environment by transforming image objects or multimedia objects etc from higher quality version to lower quality version to best suit the capability of users' devices. IBM's Image Transcoding Proxy is a prototype system to show the potentials of adaptive web content delivery [261, 262]. The main focus of the system is on the image transcoding. It can convert the quality of images to different levels based on the image purpose, network bandwidth availability and client's characteristics and preference. Mowser [220, 223] is a transcoding proxy that allows a mobile user to specify his viewing preferences, and performs transcoding of HTTP streams accordingly. When requested by end users, it can reduce the size or color for image objects, or select and transmit representative frames of video objects to the user. Other example transcoding systems include Pythia, TranSend [219], InfoPyramid [263, 264], Transcoding Publisher [265], and AppCelera ICX [215] etc.

Redline Networks's Real-Time Acceleration Appliance employs content optimization technology which cuts out non-renderable data from HTML files to decrease their retrieval latency [224]. A number of other systems from FileNET, Web Site Optimization, WebTrimmer, HypnoText etc also provide similar feature [266, 267, 268, 269].

As to peer-to-peer web system, such techniques are mainly exploited for web hosting system currently. IBM runs an experimental peer-to-peer web hosting system named YouServ [243, 244]. YouServ makes web serving more effective to wide web community by improving the availability, scalability, latency etc. for web services. Another example is the BadBlue's web server, which is also a full-blown web server based on P2P file-sharing techniques [270].

2.5 Summary

In this chapter, we reviewed the protocols and languages related to web content

delivery and the major research works in the area of web acceleration. Web caching is the first major technique used to tackle the performance problem of web retrieval. Various aspects of caching have been extensively studied, and it has been shown that caching is an effective mechanism in improving web retrieval latency, although its performance is limited by some factors. The effectiveness of caching spawns other caching-based mechanisms such as CDNs and prefetching. Most works on prefetching focus on finding good prediction algorithms which generate high accuracy of prediction and little extra network traffic. But as the prediction of web users' requests is a very tough task, there is no outstanding and convincing algorithm being found yet. As the limitation of caching-based mechanisms being realized, researchers are looking into a new direction which tries to accelerate the downloading process of web retrieval. We also surveyed the major works in this direction which include persistent-connection, pre-connection, bundling, encoding, transcoding, content optimization and selection, parallel fetching and peer-to-peer web system etc. We also examined some typical academic and commercial web acceleration systems available today.

Although the studies in web acceleration are extensive, there are still some open issues. Firstly, page retrieval latency is not given enough attention. Most existing studies are based on object level information, which is insufficient and sometimes inaccurate. Secondly, operation and chunk level details are not well studied. Thirdly, we still lack of a good model to capture web retrieval process accurately. Although there are quite a number models proposed such as those in [271, 78, 272, 273, 106, 274, 275] etc., they are either too coarse or not appropriate for capturing page retrieval. Finally, it is still preferable to look for effective acceleration mechanisms which have special emphasis on reducing page retrieval latency. We will study these aspects of

web content delivery in the rest parts of this thesis.

Chapter 3 Cacheability of Web Objects

3.1 Introduction

World Wide Web has been the major service on the Internet [1]. As the scale of web and the number of its users continue their exponential growth, the problem of congested network become more and more serious and users often experience long latency when surfing the web. To alleviate the problem, web caching was introduced and has been widely used in the current web system [10].

A web cache is a server which usually lies in front of a local area network (LAN) and connects users in the LAN to web servers on the Internet. The primary function of web cache is to retrieve web objects on behalf of the users and serve users with the objects that they requested. As web cache also keep copies of retrieved objects in its local storage, subsequent requests may be served locally if the requested objects can be found in the local storage. Therefore, user-perceived latency can be reduced significantly as the latency for fetching of the object from remote servers is eliminated.

(a) Hit-Fresh (b) Hit-Stale

Figure 3.1 Two situations of cache hit

Obviously, the reduction of latency can only be achieved when a request is hit in web cache. In reality, the reduction could vary greatly due to the complex situation in real systems. Because objects stored in web cache have their times to live (i.e. TTL), they may not be able to be reused directly. Taking this into consideration, the Cache-Hit of a request can be classified into two categories (see Figure 3.1):

a. *Hit-Fresh*: The cached object is still in its TTL and is considered the same as the one on the origin server. In this case, web cache can serve user requests with this copy without validation with the origin server.
b. *Hit-Stale*: The cached object has expired beyond its TTL. Web cache must communicate with the origin server (or an up-stream web cache) to find out whether the object has changed before it can use the local copy to server a new request. The response from the origin server will be either a "Not-Modified" message in the case that the object is not changed, or a new copy of the object in the case that the object has been modified.

Between these two Cache-Hit categories, *Hit-Fresh* can surely reduce network latency because no transmission of the data between web cache and origin server needs to be performed. However, for *Hit-Stale*, web cache will have to communicate with the origin server. Regardless whether the server's reply is a new copy of the object (when the object has been modified) or a "Not-Modified" message (when the object is not changed), the network latency incurred is often comparable to that of a cache miss. This is because the "Not-Modified" message will incur at least one chunk of data being transferred from server to web cache. We conducted experiments to study the ratio of first chunk latency vs. whole object latency. The result is shown in Figure 3.2. The graph shows that for the majority of objects, the first chunk latency occupies more than 90% of the whole object latency. On average, the ratio is about 78%, which is very significant. The reason for this high ratio is mainly because web retrieval time consists of some components independent of content size, such as the server location resolution time and connection establishment time etc. This result indicates that validation requests are almost as costly as normal retrieval requests. So, although *Hit-Stale* is also cache hit, it often incurs latency comparable to that of cache miss. In order to get the

best benefits from web caching, it is important to not only make as many as possible web objects to be cacheable, but also make the TTL of cached objects as long as situation permits to minimize the necessity of doing validations.

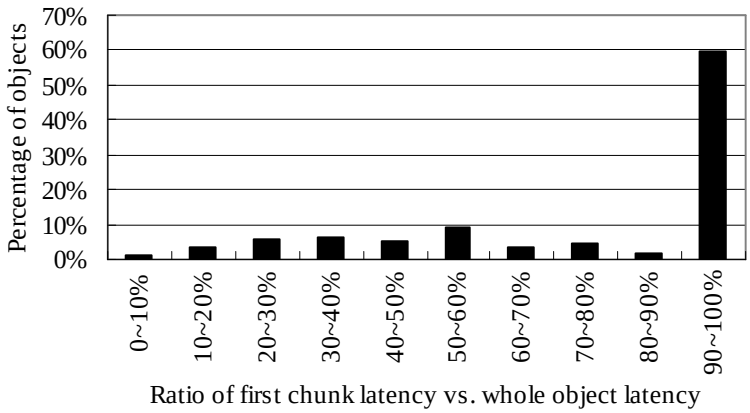


Figure 3.2 Distribution of first chunk latency vs. whole object latency

The cacheability of web objects, which is the availability and duration that web objects can be kept in a web cache, is controlled by some factors (mainly HTTP headers) which come along in the responses of objects from web servers. To decide whether an object is cacheable or not, web caches typically examine certain factors in a pre-defined order and make the decision based on the first satisfied factor. In reality, the response of an object often contains multiple factors. Therefore, to simply improve one factor may not result in improvement in cacheability because web caches may then meet other factors which also appear in the response and make decision based on them. The duration that an object can be kept in web cache and still considered fresh, i.e. the TTL of an object, is also controlled by some HTTP headers which are provided by web servers. The values of these headers are supposed to be set in accordance with the properties of web objects. However, we found in our study that these essential headers are often assigned inappropriate values by web servers. This results in considerable performance loss in web caching.

Previous studies on web caching mainly focus on algorithms of caching and

prefetching [30, 114, 115, 76, 195, 129, 137, 89, 86, 141]. The works on cacheability controls are very limited. Some relevant works studied errors in timestamp-based HTTP header values [113], cacheable reasons [111], and freshness controls [81] and the age penalty in hierarchical cache system [153], etc. These works lack of studies on the effect of multiple-factor co-occurrence and the accuracy of current settings for TTLs of objects. In this chapter, we would like to dig into the relationship among the co-occurrent factors to reveal the effectiveness of them in the co-occurrence situation, and investigate the accuracy of the settings for the TTLs of objects to reveal its impact on web caching. The results revealed in this study would help in improving web caching performance and bandwidth utilization by making more objects to be cacheable and cached longer.

In this chapter, the TTL refers to the time period used by web caches to decide whether an object is fresh, while lifetime of an object refers to the time difference between two consecutive changes of the object content.

The rest of this chapter is organized as follows. In Section 3.2, we reveal the factors and algorithms for determining cacheability of objects by studying HTTP protocol and a real web caching system Squid. Section 3.3 and 3.4 discuss our methodology and results of this study. Section 3.5 concludes this chapter.

3.2 Study of Cacheability Algorithms

In essence, the availability and the TTL of web objects for caching are mainly controlled by certain HTTP header directives found in the responses from web servers. We studied in detail the HTTP protocols to understand how these header directives are used to decide the cacheability of web objects. We also systematically examined a real web caching system, namely Squid [73], to get a better idea about how these header directives are used in real web caching systems. The version of Squid that we used in

our study is 2.4.STABLE3, which was the latest version as at the time of our study.

In this section, we study the algorithms and factors for determining the cacheability of web objects in two aspects. First, how a web object is determined to be cacheable is examined. Then we studied the algorithm for determining the TTL of web objects.

3.2.1 Algorithm and Factors for Cacheable and Non-cacheable

In general, whether a web object is cacheable is determined by the presence or absence of certain HTTP headers and the different status codes of HTTP responses. Table 3.1 lists the main HTTP response headers that are related to caching. These headers are selected based on the specification of HTTP/1.0 and HTTP/1.1 and the implementation of Squid.

Table 3.1 HTTP headers that related to cacheability of web objects

Header Name	Usage
Age	Specify the age of response entity since the time the response was generated by the origin server
Authorization	Pass user's authentication credentials to origin server
Cache-Control	Control various aspects of caching
Content-Length	Specify length of entity object in bytes
Content-Type	Specify media type of the object
Date	Indicate date and time at which the message was generated
Expires	Specify expiration date and time of object
Last-Modified	Specify creation or last modification time of object on origin server
Pragma	This header is being phased out in favor of the Cache-Control header
Vary	Lists request headers on which document content may vary

The HTTP response status codes can be classified into 4 classes for deciding cacheability of objects. Table 3.2 gives these classified status codes of HTTP responses. If the response of an object contains status code belonging to Class 4, it is deemed as non-cacheable. If the response's status code belongs to Class 3, the object can be negatively cached for some time. For this type of objects, the responses from servers are actually error messages. If a web cache receives new request for such objects in the near future, it will assume the same error will happen and it will simply reply the request with the negatively cached object. For objects with status code belonging to

Class 1 and 2, they are possibly cacheable. The final decision whether it is cacheable or not is further determined by if it can be validated. Web cache will not cache an object which can not be validated at a later time. Whether an object can be validated is also determined by some HTTP headers such as “Expires”, “Last-Modified” and “Content-Length” etc. If there is no such headers in presence or their values are inappropriate, then the objects will be considered as non-cacheable. (Note here that the “Content-Length” header is used in a reverse manner. If the value of this header is zero, then the object will be considered as non-cacheable because there is no use to cache a zero-byte object.)

Table 3.2 Classified status codes of response

Class	Status Codes
Class 1	200(OK) 203(Non-Authoritative Information) 300(Multiple Choices) 301(Moved Permanently) 410(Gone)
Class 2	302(Moved Temporarily)
Class 3	204(No Content) 305(Use Proxy) 400(Bad Request) 403(Forbidden) 404(Not Found) 405(Method Not Allowed) 414(Request-URI Too Long) 500(Internal Server Error) 501(Not Implemented) 502(Bad Gateway) 503(Service Unavailable) 504(Gateway Timeout)
Class 4	206(Partial Content) 303(See Other) 304(Not Modified) 401(Unauthorized) 407(Proxy Authentication Required) Other codes and Invalid codes

We categorize the conditions for determining cacheability of objects into two sets of factors. The first set of factors contains 12 factors which will rule that an object is non-cacheable. The second set of factors is for making cacheable decision and there are 4 factors in this set.

The 12 factors for non-cacheable are listed in Table 3.3. These factors are usually checked in the order as shown in the table. When one factor is found satisfied, the rest of the factors will not be checked, even if there are still more factors in the HTTP response. Note that the response of an object may contain more than one factor, but it will not have all the factors.

If an object passes the check of the factors listed in Table 3.3 and no factor is

found satisfied, then it is possibly cacheable. The final decision whether it is cacheable is further decided by some other factors which are listed in Table 3.4. These factors are also often checked in the order as shown in the table. Again, the response of an object may contain more than one factor, but when a factor is found to be satisfied, the rest of the factors will not be checked.

Table 3.3 Factors for non-cacheable

Factors	Description
fn1	There exists the header “Cache-Control: private”
fn2	There exists the header “Cache-Control: no-cache”
fn3	There exists the header “Cache-Control: no-store”
fn4	There exists the header “Vary”
fn5	There exists the header “Pragma: no-cache”
fn6	There exists the header “Content-Type: multipart/x-mixed-replace”
fn7	Status code belongs to Class 4
fn8	Status code belongs to Class 1, and server specified “must-revalidate”
fn9	Status code belongs to Class 1, and the object cannot be revalidated because there is no “Last-Modified” header
fn10	Status code belongs to Class 1, and the object cannot be revalidated because “Content-Length” is 0
fn11	Status code belongs to Class 1 and the object is fresh in 60 seconds or it can be revalidated, but there are no “Date”, “Last-Modified” and “Expires” headers
fn12	Status code belongs to Class 2, and there is no “Expires” header

Table 3.4 Factors for cacheable

Factors	Description
fc1	Status code belongs to Class 1, and there exits “Date” header
fc2	Status code belongs to Class 1, and there exits “Last-Modified” header
fc3	Status code belongs to Class 1, and there exits “Expires” header
fc4	Status code belongs to Class 2, and there exits “Expires” header

3.2.2 Algorithm for TTL

The factors discussed in the previous subsection are used to determine whether a newly retrieved web object is eligible to be kept in cache. For the web objects already in cache, there is an issue of checking their freshness when they are requested again. A web object is considered fresh only when its associated TTL has not expired. Only fresh objects can be used to serve a new request directly. Otherwise, a validation process between web cache and origin server has to be carried out.

As it was mentioned, the TTL of a web object is often deduced from some HTTP response headers. Based on the HTTP protocol and the implementation of Squid, the

algorithm for deciding the TTL of objects can be outlined as follows:

1) Check the directive of “Cache-Control” header

   If a cached web object has a header of “Cache-Control: proxy-revalidate | must-revalidate”, it can not be used to serve a new request without validating its freshness with origin server.

2) Check the value of “Expires” header

   If the cached object has an “Expires” header and its value shows that the object is still valid at the time when the checking performed, the cached copy will be considered as fresh and no validation with origin server is needed. Otherwise, the local copy must be validated with origin server before it can be used again.

3) Local heuristics based on object age

   Every web object in cache has an age value associated with it. This age may be calculated based on the headers like “Date” or “Cache-Control: max-age”. If this age is greater than a predefined maximum age (e.g. three days), the local copy of the object will be considered as stale and it must be validated with origin server before being used again.

4) “Last-Modified” factor algorithm

   If the cached object has a “Last-Modified” header, a “stale age” is calculated based on it. The value of “stale age” is a fraction of the time difference between the time at which the object is stored in cache and the time specified by the “Last-Modified” header. If the age of the cached object is smaller than this “stale age”, the local copy will be considered as fresh and no validation is needed. Otherwise, it must be validated before being reused.

5) Local heuristics based on object age

   If the age of the cached object is smaller than a predefined minimum age, the local

copy will be considered as fresh and there is no need to do validation. Otherwise, validation is needed. However, the predefined minimum age is often set to zero, so the object will always be determined as stale when the decision control reaches here.

From the above study, we can see that the headers “Cache-Control”, “Vary”, “Pragma”, “Expires”, “Last-Modified”, “Date”, and “Content-Length” etc are very important in determining the cacheability of web objects. The presences of these headers and proper values for them have great impact on the cacheability of web objects. We conducted trace-driven simulations to study the effectiveness of the factors and the accuracy of the TTL settings. Our results reveal which factors are the most important factors in determining the cacheability of objects and how TTL settings can be improved to further improve the web retrieval performance.

3.3 Methodology and Test Set

We did trace-driven simulations to investigate the effectiveness of the cacheability factors and the accuracy of the TTL settings in current web system. The trace we used for our experiments was from the National Laboratory for Applied Network Research (NLANR) [276]. NLANR’s hierarchical proxy system adopts Squid proxy caching software, which is the same software as we used to study the algorithms. NLANR publishes about nine traces on their server daily. One trace dated 12th March 2002 is randomly chosen for our experiments. The trace contains about 1.36 million requests.

Our experiments rely much on the header information of HTTP responses, but raw NLANR traces do not have this information. In order to get such information, we implemented utilities to get HTTP response headers for all the URLs logged in the raw trace. We performed this gathering of header information only a few days after the log

date, so the header information we obtained should be very close to the actual values that would have been obtained at the time when the original trace was logged.

After HTTP header information for the trace has been got, it was fed into our simulators together with the original trace. We implemented simplified Squid-like caching and cacheability-checking algorithms in our simulators.

3.4 Results and Analysis

We first obtain some statistics about the trace. Out of more than 1.36 million requests in the trace, about 0.63 million requests are duplicated ones. This results in a maximum cache hit ratio of about 46.2%. This cache hit ratio depicts the benefit of adopting web caching.

3.4.1 Cacheability Factors

In previous section, we examined the factors for non-cacheable and cacheable as shown in Table 3.3 and Table 3.4. In this section, we would like to investigate the correlation-ship among the factors and how it affects the effectiveness of the factors. The results give us hints on which are the most important factors to improve.

3.4.1.1 Study of Factors for Non-Cacheable

Factors for non-cacheable refer to those which are used in web cache to rule that an object is non-cacheable. To improve such factors means to remove them from occurring.

In the trace, there were about 0.7 million unique objects. Our simulation finds that about 38.4% of them are non-cacheable. Figure 3.3 plots the occurring frequencies of non-cacheable factors. From this graph, we see that factors fn7, fn9 and fn12 all occur about 25% and above, while factors fn1, fn2 and fn5 also have considerable contributions.

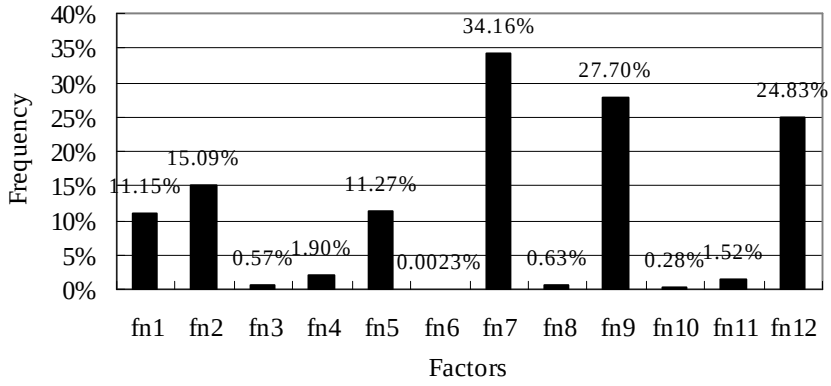


Figure 3.3 Frequencies of non-cacheable factors

We noticed that the factors for non-cacheable often does not occur alone in responses of objects. In our trace, we found that more than 40.1% of the non-cacheable responses contain multiple factors. Table 3.5 gives the top 30 factor combinations according to their occurring frequencies. We see that many of them contain more than one factor. In other words, a factor often goes with other factors in HTTP responses.

When investigating the effectiveness of the factors, we should take into consideration the effect of this kind of multi-factor co-occurrence. This is because: in the situation of multi-factor co-occurrence, the performance of web caching will not be improved by improving just one factor since other factors will then come into effect, which would still result in non-cacheable decision for objects.

Table 3.5 Top 30 non-cacheable factor occurrences

Frequency	Factors	Frequency	Factors	Frequency	Factors
116322	fn7	3371	fn1 fn12	1279	fn5 fn7
77037	fn12	3311	fn2 fn9	1015	fn5
65857	fn9	3028	fn1 fn2	944	fn1 fn2 fn9
17122	fn2 fn5	2866	fn1 fn2 fn12	910	fn2 fn5 fn8
15456	fn1 fn9	2446	fn1	625	fn10
7601	fn2 fn5 fn9	2183	fn1 fn5 fn9	526	fn1 fn2 fn5 fn9
5344	fn9 fn11	1748	fn2 fn5 fn12	450	fn2 fn4 fn9
5072	fn2	1590	fn4 fn9	440	fn2 fn3 fn5 fn7
3789	fn4	1317	fn1 fn7	396	fn1 fn5
3716	fn5 fn9	1309	fn2 fn12	382	fn1 fn2 fn5 fn12

Figure 3.4 plots the effectiveness of non-cacheable factors against their respective occurring frequencies. The effectiveness is got by taking the multi-factor co-occurrence effect into consideration. From this graph, we see that factors'

effectiveness is generally different from their respective frequencies. This indicates that the effectiveness of the factors is indeed affected by the co-occurrence relationship among them. In other words, the multi-factor co-occurrence relationship changes the curve of the graph. (Note: In Figure 3.4, the "effectiveness" is often lower than the "frequency". Here is a point to help you understand the reason: The "frequency" of every factor can be as high as 100%, while 100% is the sum of the "effectiveness" of all factors.)

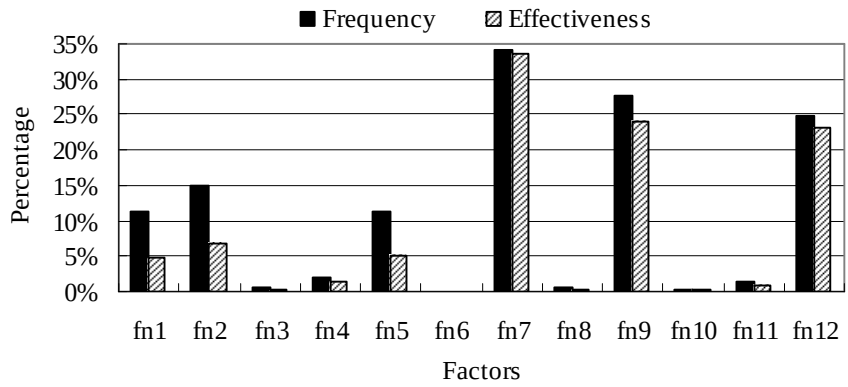


Figure 3.4 Frequencies and effectiveness of non-cacheable factors

In general, the absolute value of the effectiveness of a factor is lower than its frequency. This is because factors often occur in groups. If all factors always occur alone in HTTP responses, then their effectiveness would be the same as their respective frequencies. However, if a factor *fnx* often occurs together with other factors, its effectiveness will be lowered. This is because: other factors in the factor combination will still make the object non-cacheable when the factor *fnx* is removed. So, to occur in multiple-factor groups would lower the effectiveness of a factor.

In Figure 3.4, we see that the effectiveness of some non-cacheable factors is lower than their respective frequencies more significantly than others. Further study reveals the reason being that different factors occur in different number and different size of factor combinations. If a factor occurs more often in groups or occurs in larger factor groups, then its effectiveness will be lowered more significantly.

Figure 3.5 plots the relative distribution of "occur alone" and "occur in group" for each factor. We see that the majority of the occurrences of factors fn1, fn2 and fn5 are occurring in groups. This significantly lowers their effectiveness. As a contrast, the majority of the occurrences of factors fn7, fn9 and fn12 are occurring alone. So their effectiveness is not lowered that much by the multi-factor co-occurrence relationship. However, since these three factors do have some co-occurrence situations, their effectiveness is also lowered a little. But when compared with other factors which are affected by the multi-factor co-occurrence relationship heavily, the relative importance of these three factors even increases. So, when we put all factors together and compare their effectiveness and occurring frequencies, we see that the effectiveness of factors fn7, fn9 and fn12 is getting more significant than their frequencies do.

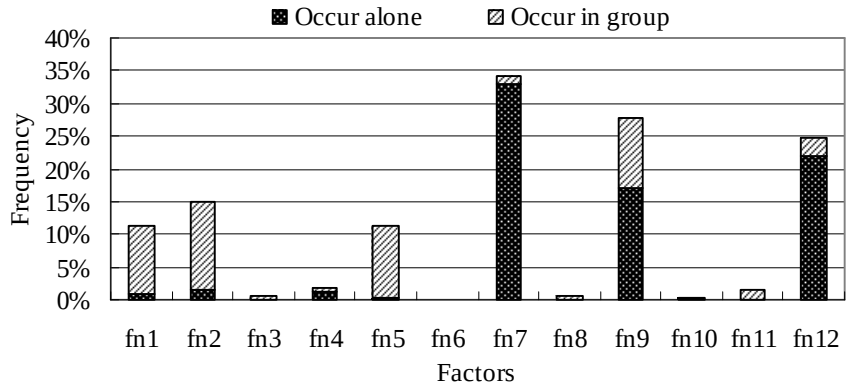


Figure 3.5 Relative distribution of "occur alone" and "occur in pair" of each factor

On the other hand, the size of the factors group (i.e. the number of factors in the group) that a factor occurs in also has impact on the factor's effectiveness. The bigger the size of the group is, the lower the effectiveness of each factor in the group will be.

Figure 3.6 plots the relative distribution of occurrence in different sizes of groups for each factor. For factors fn1, fn2 and fn5, we see that a large percentage of their occurrences are in groups with 2, 3 and 4 factors. This results in significant reduction in their effectiveness. As to factors fn3, fn8 and fn11, although a large percentage of their occurrences are in even bigger groups (up to 5-factor group in our experiments),

the impact on their effectiveness is not obvious because their relative value is too small.

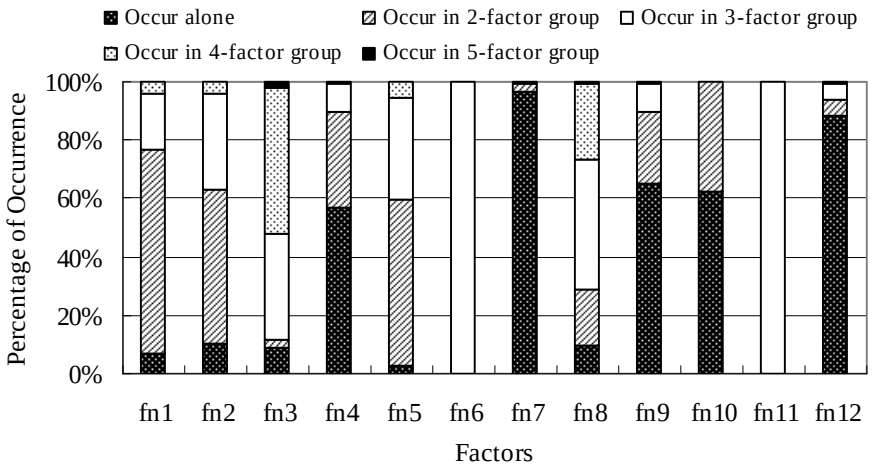


Figure 3.6 Distribution of occurrence in different sizes of groups of each factor

From the above analysis, we see that fn7, fn9 and fn12 are the most important factors for non-cacheable, and their effectiveness is more significant than their respective frequencies. Simply from the occurrence frequencies, these three factors seem to occupy about 86.7% of all HTTP responses. But when we take the multi-factor co-occurrence relationship into consideration, we find that these three factors actually contribute about 80.7% to all non-cacheable decisions. While the situation for fn1, fn2 and fn5 is that they seem to occupy about 37.5% of all HTTP responses, yet they only contribute about 16.4% to all non-cacheable decisions. By taking the multi-factor co-occurrence relationship into consideration, we see that the importance of fn7, fn9 and fn12 becomes more significant.

Refer back to Table 3.3, we see that fn7 stands for those responses which have status codes belong to Class 4, which is mainly for partial content, validation requests, redirection and authentication etc. To improve this factor may require protocol support to provide mechanisms to cache such responses. For the factor fn9, the non-cacheable decision is mainly caused by the absence of “Last-Modified” header. This can be

improved by content providers to properly configure their servers to let them provide this essential header. As to fn12, it is for “Temporarily Moved” objects. Content providers can improve this situation by quickly completing the moving of the objects and provide latest valid URLs for them.

For factors fn1, fn2, fn3, fn5 and fn8, they play similar function which is to explicitly state that the object should not be cached. Normally, these factors exhibit the purposeful behavior of content providers and they should be respected. To improve these factors, content providers are advised to examine their content carefully and use these explicit non-cacheable directives conservatively. According to our study (see the next section), many of non-cacheable objects do not change within a quite long time period. This suggests that the usage rate of these explicit non-cacheable directives can actually be reduced. Doing so will improve the performance of web caching without sacrificing the correctness of web content.

For the non-cacheable decision caused by factors fn4 and fn6, the responsibility mainly lies on web caching systems rather than content providers. This is because web caching systems like Squid currently do not support these factors well and they just do not cache objects with such factors. Improvement on this situation would require developing better caching systems to handle these factors properly. But as objects fall into this category is very few, less than 2% according to Figure 3.3, so the necessity of doing so is not high.

The factor fn10 represents the situation where the non-cacheable decision is caused by zero content-length. This situation is mostly caused by mistakes of content providers since a web request should not cause an object with zero length to be returned. Web content providers can improve this situation by carefully monitoring their content and configuring their servers. On the other hand, the occurrence of this

factor is very rare. So the negative impact it imposes on web caching is trivial.

The occurrence frequency of factor fn11 is also very low, only about 1.52% in our study. However, this factor reflects a serious situation in web caching where the essential headers “Date”, “Last-Modified” and “Expires” are all missing at the same time. Content providers should play their role in improving this situation by properly configuring their servers to provide these important headers for the responses generated from them.

3.4.1.2 Study of Factors for Cacheable

Factors for cacheable usually take effect after objects pass the check of non-cacheable factors. Objects will be considered cacheable if any factors for cacheable is found satisfied. As we see in Section 3.2, there are only 4 factors for cacheable. So the multi-factor co-occurrence relationship among those factors is relatively simple.

Figure 3.7 plots the occurrence frequencies and the effectiveness of cacheable factors. We see that the curve of effectiveness is very similar to the curve of frequency. This is because that the majority of the factors have similar opportunity to occur in groups, so their effectiveness is affected by similar weights.

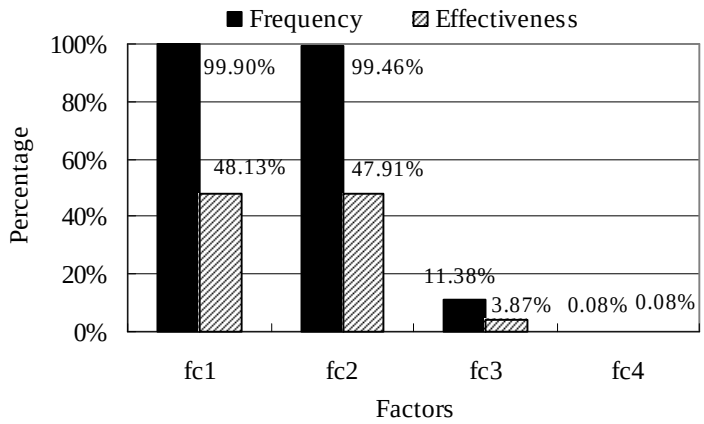


Figure 3.7 Frequencies and effectiveness of cacheable factors

Table 3.6 gives all the occurrence combinations of cacheable factors according to

their occurrence frequencies. From this table, we can see that the majority of the occurrences of the factors are in groups, only 0.09% of the occurrences have a single factor (i.e. the last two rows). Since the majority of the factors have similar opportunity to occur in groups, the impact of multi-factor co-occurrence on the effectiveness of factors is distributed quite evenly. So the relative distribution of the effectiveness of cacheable factors is quite close to the relative distribution of their respective frequencies.

Table 3.6 Cacheable factor occurrences

Frequency	Factors		
88.52%	fc1	fc2	
10.92%	fc1	fc2	fc3
0.46%	fc1		fc3
0.08%	fc4		
0.01%	fc2		

Because the number of factors for cacheable is small and the co-occurrence situation of them is simple, so the relative distribution of the effectiveness of cacheable factors is quite close to the relative distribution of their respective frequencies. But for cacheable objects, there is another important issue to study, which is the accuracy of the TTLs of them.

3.4.2 TTL Control

Cached web objects may be used to serve new user requests. This is how web cache improves retrieval latency and reduces external network bandwidth consumption. But every cached object has its TTL (i.e. time to live) in the cache. An object can be used directly only when its TTL is not expired. Otherwise, a validation communication with the origin server has to be carried out and this will cause the performance of web caching lose considerably since the latency incurred by the validation process is often comparable to those of retrieving a new object from web server.

An object's TTL is deduced based on some HTTP response headers and some heuristic algorithms. It is conceptually different from its lifetime which refers to the

time difference between two consecutive changes of the object content. The lifetime of an object is independent of TTL. It is only determined by the content of the object. In ideal situation, TTLs of objects should be set as close to lifetimes as possible so that the performance of web caching can be maximized. In other words, the TTL and lifetime of objects better to expire at the same time as much as possible.

In this subsection, we would like to investigate the discrepancy between objects' lifetimes and TTLs under today's web system settings.

In order to investigate the difference between TTL and lifetime of objects, we implement utilities to request objects and verify their TTL and content change automatically. For every newly requested object which is cacheable, we verify its content change before, at and after its TTL to see if the TTL and lifetime of objects expire at the same time. About the saying "at the same time", it might be too strict to require the lifetime and TTL of an object to expire exactly at the same point of time. In our experiments, we loose this restriction by using a small range of time instead of a single point of time: If the lifetime of an object expires at time t_i, where $TTL - \delta \leq t_i \leq TTL + \delta$, then we would regard that the lifetime and TTL of the object expire at the same time. In other words, the setting of the TTL would be considered to be accurate when this situation happens. As for the value of δ , we set it to be 5% of the TTL for TTLs longer than or equal to 1 minute. For objects with TTLs shorter than 1 minute, we exclude them from our experiments because such objects are often not cached by web caches [73].

To study the accuracy of TTL of objects, we verify if object content changes before $TTL - \delta$, between $TTL - \delta$ and $TTL + \delta$, and after $TTL + \delta$, as shown in Figure 3.8. For the after $TTL + \delta$ case, we conduct the verification up to $3 \times TTL$ away if the object content is not changed. We did not do it further because of time limitation since

the TTLs of some objects can be quite long.

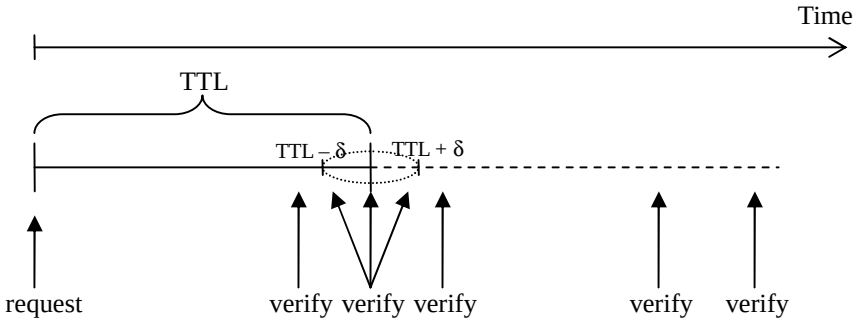


Figure 3.8 Verifying difference between TTL and lifetime

The results of the accuracy of TTLs are shown in Table 3.7. The percentage values shown in the table all refer to the whole object set used in our experiment. From this table, we see that very few (about 1.32%) objects change right within the small time range around the TTL. The majority of content change happens at times after TTL + δ. Furthermore, a rather high percentage (76.4%) of objects are unchanged even after 3 × TTL. This indicates that the TTL settings for most objects in current web system are often too conservative. This situation not only results in performance loss in web caching because many objects can not be reused even if their content is not changed, but also it imposes excessive load on the network and web servers by triggering unnecessary requests for revalidation and retrieval.

Table 3.7 Accuracy of TTL

Time	Objects changed	Objects NOT changed
$t < \text{TTL} - \delta$	0.73%	99.27%
$\text{TTL} - \delta \le t \le \text{TTL} + \delta$	1.32%	97.95%
$\text{TTL} + \delta < t \le 3 \times \text{TTL}$	21.55%	76.4%

We also noticed that there are some objects which have their content changed before the specified TTL time. In this situation, the TTL settings are set too aggressive. Although the percentage of such objects is diminutive, this situation is highly undesirable because it may result in stale and incorrect object content to be delivered to web users.

From the above results, we see that the performance of web cache can be improved greatly by configuring web servers properly to make them provide TTL headers with proper values. According to the algorithm shown in Section 3.2, the TTL of web objects is first defined by a few explicit TTL headers such as “Expires”, “Cache-Control: max-age”. Then, if these headers are absent, a heuristic algorithm will be used to calculate the TTL. Such heuristic algorithm usually does the calculation based on “Last-Modified” header and the result of it is not authoritative. Therefore, a good web server should avoid relying on the non-authoritative TTL headers like “Last-Modified”. Instead, it should use the explicit TTL headers (such as “Expires”) and provide proper values for them. According to Figure 3.7, the “Expires” header can be found in only about 11.38% of the cacheable objects. So, there is much room for improvement in this aspect.

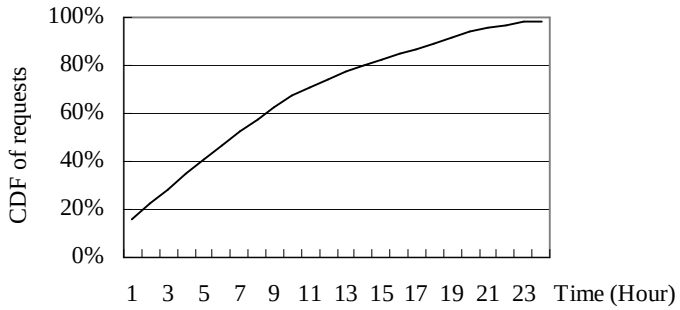


Figure 3.9 Cumulative distribution of intervals of repeated requests

Therefore, the next systematic issue to be tackled is what value would be appropriate for TTL headers. Previous results show that a great percentage of objects actually do not change at the specified TTL and even after 3 × TTL. This encourages web servers to set longer TTL for their objects. To find the proper value for TTL headers, we studied the distribution of interval of user requests for the same object and found that about 50% of cache-hit requests arrive within 6 hours’ time and about 75% arrive within 12 hours’ time (see Figure 3.9). This means, validation process for as much as 50% and 75% of cache-hit requests can be avoided if the TTL of web objects are set

to 6 hours and 12 hours respectively. Note that these TTLs are not only meant for HTML pages, they are also for embedded objects of web pages such as images. While these values may be too big for some dynamic content, they are moderate for most of embedded objects. For static-content elements, even much longer TTL values can be given.

Finally, we would like to complete this study by also investigating the accuracy of TTL settings for non-cacheable objects. There are two categories of non-cacheable objects. The first category is the objects that are set to be non-cacheable by explicit HTTP headers such as “Cache-Control: no-cache”. We consider this situation to be purposeful behavior of content providers, so that the explicit non-cacheable settings should be respected. The second category is the objects without explicit HTTP headers for non-cacheable but determined to be non-cacheable by web caches because of the absence or improper values of certain headers. From another point of view, we may regard the TTLs of the objects in this category to be less than or equal to zero, as they expire immediately when they enter web caches. Here we would like to investigate if the lifetime of the objects in the second category goes along with their TTLs.

We monitor the objects at some time intervals to see if the content has been changed. Figure 3.10 plots the result we observed. Surprisingly, we see that a lot of non-cacheable objects actually do not change in a fairly long time period. For example, after 8 hours from the first access, about 83% of the non-cacheable objects are still identical to their old versions. This suggests that the non-cacheable decision for these objects is inappropriate. The lifetime of most of these objects is actually much bigger than zero, while their TTLs are deemed to be less than or equal to zero in web caches. The causes of this situation are mainly due to the ill-configuration of web servers. If content providers can configure their servers properly to improve this discrepancy

between lifetime and TTLs of objects, significant gains in the performance of web caching can be expected. Section 3.4.1.1 has discussed the measures on how to improve the situation in this aspect.

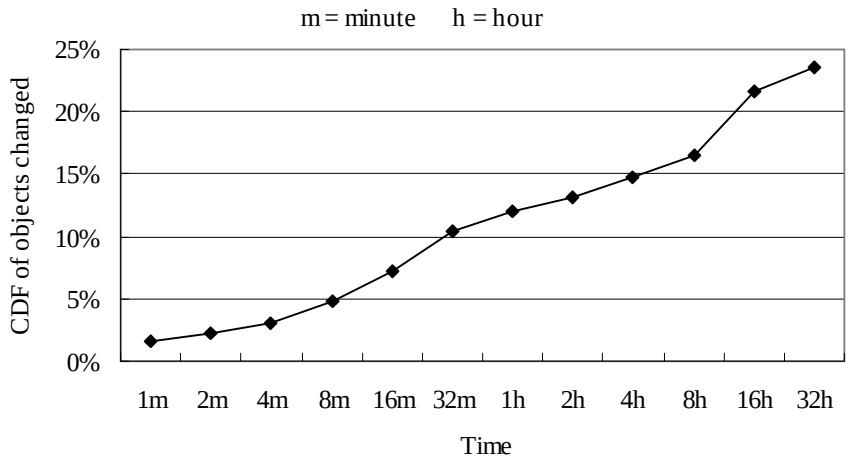


Figure 3.10 Cumulative distribution of changed objects

3.5 Conclusion

In this chapter, we systematically studied the factors affecting the cacheability of web objects. We dig into the relationship among co-occurrent factors and reveal the effectiveness of the factors in the multi-factor co-occurrence situation. The accuracy of the settings for the TTLs of objects is also investigated and our results show that TTLs for most objects are set too conservative, which results in considerable performance loss for web caching. Our study revealed the effective factors and proper settings for TTL. By improving them, considerable improvement on the performance of web caching can be expected.

Chapter 4 Web Retrieval Dependency Model

4.1 Introduction

Web retrieval latency is one of the most important issues in web content delivery. A lot of works have been done in order to improve web retrieval latency. In current web system, the basic unit of web browsing is web page, which is often made up of multiple web objects. Since web page is the basic unit of browsing, page retrieval latency would be more meaningful than object retrieval latency. However, most researchers study web retrieval latency based on object retrieval latency [30, 31, 32, 77]. We would like to point out that this method would result in inaccurate results about web retrieval latency because there is actually complex relationship between the retrieval processes of objects in a page, which prevents object retrieval latency from being translated directly into page retrieval latency. For example, the triggering of the requests for the inline images of a page is dependent on the retrieval of the HTML file of the page. When the triggering of the requests would happen will not be able to be accurately identified unless we go into more detailed level than the object level. Besides, most common web browsers often fetch multiple objects in parallel. All these complicate the mapping of object retrieval latency into page retrieval latency. Page retrieval latency can not be computed as the simple sum-up of objects' retrieval latency. To well understand and study the complex inter-relationship affecting web retrieval latency, we will need a more precise model at more detailed level. In this chapter, we propose a detailed operation-level Web Retrieval Dependency Model (WRDM) to study web retrieval latency. We show that our model reveals/captures some properties of web retrieval which can not be seen at object level.

Before we give the description of our model, it is necessary to have some basic understanding on the details of retrieval processes for web pages and objects.

In current web system, a web page is often made up of multiple objects. Among the objects in a page, there is one primary object corresponding to the URL of the page. This object is generally an HTML file (or script files like .asp files) which contains a number of URLs specifying some other objects needed by the web page. We call this primary object *Container Object (CO)* in our study. For those objects whose URLs are *defined* in the CO, they are referred to as *Embedded Objects (EO)* of the page. The most commonly seen EOs are inlined images. The content of CO and the EOs are interpreted and displayed together to render the full view of a web page.

Generally, the retrieval process for a web page starts with the submission of a request which comprises information about request method, URL address and some request headers. This URL address identifies the CO of the web page only. Following the submission of a request, the location of the web server is resolved and a network connection between client and the server is established. Then the request message is transferred to the server. Upon receiving a request, a web server will reply the request with HTTP response headers and the data of the requested object. The reply message is streamed to the client through the same network connection in a sequence of network packets, which are seen and referred to as *data chunks* in the HTTP-level web system. When a chunk of data reaches the client, the content of the chunk will be interpreted by user's web browser. The results of this interpretation of a data chunk typically include caching action, displaying the content in user's web browser window, as well as triggering further retrieval processes for EOs if there are URLs defining them in the data chunk. For all the subsequent requests for EOs, the replies are also delivered and interpreted in a chunk-by-chunk way. When all the objects belonging to a page are retrieved, resources such as network connection occupied by the requests are released. In some cases, a retrieval process may be prematurely interrupted at any stage. When

such interruption occurs, the resources occupied by the requests will also be released.

In a web page, there usually exist a number of hyperlinks which are actually URLs for new web pages. If such a hyperlink is clicked by a user, the retrieval process for a new web page corresponding to the clicked hyperlink will be initiated and the above procedure will repeat.

4.2 Web Retrieval Dependency Model (WRDM)

The retrieval process for objects and pages involves a sequence of operations such as location resolution, establishment of network connection, and data chunks transfer etc. We propose a *Web Retrieval Dependency Model* to capture the retrieval process at detailed operation level. The idea of our model is to map the relations among all operations of the retrieval process into a *directed graph*. We symbolize each operation in a retrieval process by a *vertex* of a graph. Then since the relationship between two operations can be regarded as a precedence requirement, it can be represented by a *directed arc* connecting the two vertices associated with the operations involved. The resulting graph is called *Web Retrieval Dependency Graph (WRDG)*.

In a WRDG,

- A *vertex* represents the completion state of an operation in the retrieval process of an object (either CO or EO). Some operations represented by vertices may have certain information associated with them.
- An *arc* connecting two vertices represents the precedence relationship between the two operations represented by them. This precedence relationship is often referred to as dependency between operations.
- Each arc carries a *weight* which represents the time spent in completing the operation represented by the target vertex. The time is measured from the

completion point of the operation represented by the source vertex to the completion point of the operation represented by the target vertex.

Below we give the precise definitions of the key terms used in the WRDM model. WRDM model can capture the characteristics of web retrieval process at various levels. In our study, we consider three levels of web retrieval process:

1) Page-level (or inter-page level)

   This level captures the retrieval process among pages;

2) Object-level (or intra-page level, inter-object level)

   This level captures the retrieval process among objects in the same page;

3) Chunk-level (or intra-object level)

   This level captures the operations on data chunks and other resources in the retrieval process of a single object. Because the basic unit of data size streaming from web server to client at the HTTP level is chunk, so we use it to refer to the operation details at HTTP level.

We introduce three indices, k , i and j to index the entities encountered in each of the three levels respectively.

Definition 4.1: Page Index *k*

In WRDM, page index k is a natural number used to index pages among a set of pages that are visited by clients. The range of page index k is defined to be from 0 to p , where $p \geqslant 0$.

A page in the set of pages is represented by *Page(k)*, where $0 \leqslant k \leqslant p$.

A web page is usually made up of multiple objects (refer to the Container Object and Embedded Object concepts in previous section). We use an index, i , to index the objects in a page.

Definition 4.2: Object Index *i*

In WRDM, object index i is a natural number used to index objects that belong to the same page *Page(k)* . The range of i is relevant to the page. For a given page *Page(k)* , the range of i is defined to be from *0* to *f(k)* , where *f(k)* is the number of embedded objects belonging to *Page(k)* . In the rest part of this thesis, wherever is appropriate, we may use o to refer to this *f(k)* , i.e.:

$$o = f(k)$$

Because object index i is related to page index k, an object belonging to a page *Page(k)* is represented by *Obj(k,i)* , where $0 \leqslant k \leqslant p$ and $0 \leqslant i \leqslant o$. When it is not misleading in a particular context, we may omit the page index k and just use *Obj(i)* to refer to *Obj(k,i)* .

In terms of objects, the retrieval process of *Page(k)* can be represented by the retrieval of a sequence of objects:

$$Page(k) = \langle Obj(k,0), Obj(k,1), \ldots, Obj(k,i), \ldots, Obj(k,o) \rangle, \quad \text{where } o = f(k)$$

Among the objects in page *Page(k)* , *Obj(k,0)* is the Container Object of the page. Other objects, *Obj(k,i)* where $1 \leqslant i \leqslant o$, are Embedded Objects.

When a web object is retrieved from a web server to a client, the transfer of the object is often made up of a sequence of chunks of data. We introduce an index, j , to index the chunks in the chunk sequence.

Definition 4.3: Chunk Index *j*

In WRDM, chunk index j is a natural number used to index chunks corresponding to the transfer of an object from a server to a client. The range of j is relevant to the object. For a given object *Obj(i)* , the range of j is defined to be from *0* to *g(i)* , where the *g(i)* is the number of chunks that are returned from a web server to a client for the transfer of *Obj(i)* minus one. In the rest part of this thesis, wherever is appropriate, we will use c to refer to this *g(i)* , i.e.:

$$c = g(i)$$

As chunk index *j* is related to object index *i* and object index *i* is in turn related to page index *k*, a chunk of an object *Obj(k,i)* is represented by *Chk(k,i,j)* , where $0 \leqslant k \leqslant p$, $0 \leqslant i \leqslant o$ and $0 \leqslant j \leqslant c$. When it is not misleading in a particular context, we may omit the page index *k* and the object index *i* and just use *Chk(j)* to stand for *Chk(k,i,j)* .

For a given object *Obj(i)* , the value of *c* is not fixed. It depends on the status of the network and the workload on the web server and the client. In terms of chunks, the transfer of an object *Obj(k,i)* can be represented by a sequence of chunks:

$$Obj(k,i) = \langle Chk(k,i,0), Chk(k,i,1), \ldots, Chk(k,i,j), \ldots, Chk(k,i,c) \rangle, \quad \text{where } c = g(i)$$

Among the chunks belonging to an object *Obj(k,i)* , *Chk(k,i,0)* usually contains the response headers information of the object, and perhaps some data of the object body as well. For other chunks, *Chk(k,i,j)* where $1 \leqslant j \leqslant c$, they usually contain only the data of the object body.

Definition 4.4: Object Request

An object request is a message packet sent from a client to a server, which specifies a single object to be requested from the server. A request message packet consists of a request method, URL address of an object, protocol version number, and some headers information.

For a given object *Obj(k,i)*, the corresponding object request is symbolized as *Req(Obj(k,i))* . Note that an object request is for a single object, regardless of whether the object is a container object or an embedded object.

Definition 4.5: Page Request

Given a page *Page(k)*, its corresponding page request, symbolized as *Req(Page(k))* , is made up of a sequence of Object Requests:

$$Req(Page(k)) = \langle Req(Obj(k,0)), Req(Obj(k,1)), \ldots, Req(Obj(k,i)), \ldots, Req(Obj(k,o)) \rangle$$

where $0 \leqslant i \leqslant o$. $Req(Obj(k,0))$ is the initial request submitted for the page $Page(k)$ by a user and this request actually requests for the container object of the $Page(k)$. Other object requests, $Req(Obj(k,i))$ where $1 \leqslant i \leqslant o$, are the requests for embedded objects of $Page(k)$ and they are triggered by the interpretation of the content of the container object of $Page(k)$.

Note that $Req(Page(k)) \neq Req(Obj(k,0))$.

In some situations, an embedded object of a page could be also a container object which has its own embedded objects. For simplicity reason, we do not include such situations in our study, although our WRDM model can be extended to cover such situations.

Definition 4.6: Web Retrieval Dependency Graph (WRDG)

A WRDG is a weighted directed graph $G = (V, E)$, where V and E have the following members respectively:

- $V = \bigcup_{k=0..p} \bigcup_{i=0..o} \{ \{ \upsilon_{r(k,i)}, \upsilon_{l(k,i)}, \upsilon_{c(k,i)}, \upsilon_{s(k,i)}, \upsilon_{e(k,i)} \} \cup \{ \bigcup_{j=0..c} \{ \upsilon_{d(k,i,j)} \} \} \}$
- $E = \{ \bigcup_{k=0..p} \bigcup_{i=0..o} \{ \{ \langle \upsilon_{r(k,i)}, \upsilon_{l(k,i)} \rangle, \langle \upsilon_{l(k,i)}, \upsilon_{c(k,i)} \rangle, \langle \upsilon_{r(k,i)}, \upsilon_{c(k,i)} \rangle, \langle \upsilon_{c(k,i)}, \upsilon_{s(k,i)} \rangle, \langle \upsilon_{s(k,i)}, \upsilon_{d(k,i,0)} \rangle, \langle \upsilon_{d(k,i,g(i))}, \upsilon_{e(k,i)} \rangle \} \cup \{ \bigcup_{j=1..c} \{ \langle \upsilon_{d(k,i,j-1)}, \upsilon_{d(k,i,j)} \rangle \} \} \} \}$

  $\cup \{ \bigcup_{k=0..p} \bigcup_{i=1..o} \bigcup_{j=0..c} \{ \langle \upsilon_{d(k,0,j)}, \upsilon_{r(k,i)} \rangle \} \}$

  $\cup \{ \bigcup_{\substack{x,y=0..p \\ x \neq y}} \{ \langle \upsilon_{e(x,0)}, \upsilon_{r(y,0)} \rangle \} \}$

In addition, any member of the following E' can also be the member of E :

$$E' = \{ \bigcup_{k=0..p} \bigcup_{i=0..o} \{ \langle \upsilon_{r(k,i)} , \upsilon_{e(k,i)} \rangle , \langle \upsilon_{l(k,i)} , \upsilon_{e(k,i)} \rangle , \langle \upsilon_{c(k,i)} , \upsilon_{e(k,i)} \rangle ,$$

$$\langle \upsilon_{s(k,i)} , \upsilon_{e(k,i,0)} \rangle \} \} \cup \{ \bigcup_{k=0..p} \bigcup_{i=0..o} \bigcup_{j=0..c-1} \{ \langle \upsilon_{d(k,i,j)} , \upsilon_{e(k,i)} \rangle \} \}$$

- Here p, o and c are not fixed, and $p \geqslant 0$, $o \geqslant 0$, $c \geqslant 0$.
- For every member of E, there is a real number w associated with it. This number w is greater than or equal to zero and is called the ***weight*** of the arc.

The types of vertices found in WRDG are defined as follows.

Definition 4.7: Request Initiation Vertex

A vertex in a WRDG is said to be a request initiation vertex $\upsilon_{r(k,i)}$ if it represents the submission of a web object request ***Req(Obj(k,i))*** , where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page k and $0 \leqslant i \leqslant o$. This vertex has some associated information such as the request method, the URL address of the requested object, as well as some other request header information.

For each object, there is exactly one request initiation vertex $\upsilon_{r(k,i)}$ for the retrieval process of it.

Definition 4.8: Location Resolution Vertex

A vertex in a WRDG is said to be a location resolution vertex $\upsilon_{l(k,i)}$ if it represents the location resolution for the URL address from an object request ***Req(Obj(k,i))*** , where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page k and $0 \leqslant i \leqslant o$. This vertex has some associated information such as the IP address of the web server where the requested object resides.

In most cases, the URL in a request is in domain-name format. The location resolution for such URL is typically a DNS process and the result of such location resolution is usually the server's IP address which is the location of the server in the Internet. The location resolution vertex $\upsilon_{l(k,i)}$ donates the completion state of this DNS

process. For each object request, there is one location resolution vertex $\upsilon_{l(k,i)}$. However, this vertex may be bypassed in some situations. We will explain this further later.

Definition 4.9: Network Connection Vertex

A vertex in a WRDG is said to be a network connection vertex $\upsilon_{c(k,i)}$ if it represents the operation of establishing network connection between a client and a web server for an object request *Req(Obj(k,i))* , where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page k and $0 \leqslant i \leqslant o$. There is resource associated with this vertex, that is: network connection.

Assuming no "persistent-connection" or "pre-connection" mechanism is used, there will be one network connection vertex υ_c for each object request. In the presence of "persistent-connection" or "pre-connection" mechanism, we may still keep this vertex in the graph, only that the weight of the arc connecting to this vertex may become much smaller, up to zero.

Definition 4.10: Request Sending Vertex

A vertex in a WRDG is said to be a request sending vertex $\upsilon_{s(k,i)}$ if it represents the operation which sends out the request message of an object request *Req(Obj(k,i))* to a web server through the network connection that has been established by the network connection vertex $\upsilon_{c(k,i)}$, where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page k and $0 \leqslant i \leqslant o$.

For every object retrieval process, there will be exactly one request sending vertex $\upsilon_{s(k,i)}$. This vertex cannot be bypassed in any way.

Definition 4.11: Data Chunk Vertex

A vertex in a WRDG is said to be a data chunk vertex $\upsilon_{d(k,i,j)}$ if it represents the transfer of a chunk of data for a requested object *Obj(k,i)* from a web server to a client, where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page k and

$0 \leqslant i \leqslant o$, j refers to one data chunk in the data chunk sequence that corresponds to the transfer of the *Obj(k,i)* and $0 \leqslant j \leqslant c$. This vertex has some associated information such as the response data.

The first data chunk in an object's transfer, denoted by the vertex $\upsilon_{d(k,i,0)}$, contains the response headers information of the object, and perhaps some data of the object body as well. For the subsequent data chunks, denoted by the vertices $\upsilon_{d(k,i,j)}$, where $1 \leqslant j \leqslant c$, they usually only contain the data of the object body.

When a data chunk reaches a client, its content will be interpreted by the client's web browser. The results of the interpretation include caching action, rendering of the content in the browser window, and, in the case of container object, triggering new object requests for embedded object.

There is a special characteristic about the data chunk sequence: The order of data chunks being interpreted must be in a strict successive sequence order, while the order of them being transferred may be in any order.

Definition 4.12: Ending Vertex

A vertex in a WRDG is said to be a ending vertex $\upsilon_{e(k,i)}$ if it represents the operation of releasing resources (such as network connection) for an object request *Obj(k,i)* , where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page k and $0 \leqslant i \leqslant o$.

When the transfer of data chunks is finished or the retrieval process is interrupted, resources that have been occupied by the request will be released. Usually, this is the operation which releases the memory space and network connection. The released network connection may be either closed or handed over to a pool to keep alive for future use (when "persistent-connection" is used).

Very often, when the retrieval of objects for a page has completed or interrupted,

a user may initiate a new page request by either clicking on a hyperlink in the current page or keying in a new page URL address in the address bar of his/her web browser. In such cases, the ending vertex υ_e will also imply the initiation of the new request.

In any case, there is exactly one ending vertex $\upsilon_{e(k,i)}$ for the retrieval process of an object. When this vertex is reached, the retrieval process for the object is considered finished.

In this study, we only define the above six types of vertices in our WRDM model representing six operations in web retrieval. However, it is worth mentioning that our model can actually be altered to include more or less types of vertices/operations to cater for the needs in different situations. For example, a new type of vertex representing the operation of "access control" can be included in the graph when needed. But for the studies in this thesis, the above six types of vertices is sufficient. So we just use the above definitions.

There is precedence relationship between the operations of the retrieval process for pages and objects. Such relationship is referred to as dependency in our study. Based on the cause of the relationship, the dependencies in web retrieval can be classified into two categories: (1) Information Dependency, and (2) Happened-before Dependency. If an operation depends on some information produced by its previous operation, then the dependency between them is called Information Dependency. Otherwise, the dependency would be treated as Happened-before Dependency. One example of Happened-before Dependency is the dependency between "network connection establishment" operation and "request sending operation", i.e., the network connection has to be established before a request can be sent out.

These dependencies among operations of a retrieval process can be captured by the directed edges (which are called arcs) of WRDG. In a WRDG graph, each vertex

represents an operation in its completion state. An arc connecting two vertices represents the dependency between the two operations represented by the two vertices.

Before giving the definitions of the types of arcs that a WRDG may have, we first give the definition of the weight of an arc so that we can describe the meaning of the weight for each arc when we give the definition of the arcs.

Definition 4.13: Weight of an Arc

In a WRDG, the weight of an arc $\langle \upsilon_x , \upsilon_y \rangle$ stand for the time spent in completing the operation symbolized by the vertex υ_y , the timing starts from the completion state of the operation represented by the vertex υ_x , where υ_x and υ_y are two different vertices defined above.

Now we are ready to move on to the definitions of arcs. A WRDG graph can have the following types of arcs.

Definition 4.14: Location Resolution Arc

An arc in a WRDG is said to be a location resolution arc $a_{l(k,i)}$ if it connects a request initiation vertex $\upsilon_{r(k,i)}$ to a location resolution vertex $\upsilon_{l(k,i)}$:

$$a_{l(k,i)} = \langle \upsilon_{r(k,i)} , \upsilon_{l(k,i)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page and $0 \leqslant i \leqslant o$. The weight of this arc denotes the time spend in resolving the location for the web server appeared in the URL address of the object request $Req(Obj(k,i))$. In most cases, this is the time for DNS process and the resolved location is usually IP address.

In some cases, the address of the web server appeared in a URL is already in the form of numeric IP address, eg. http://137.132.12.124/index.htm. For such URLs, the location resolution operation for them will finish immediately upon identifying the numeric IP address. In these cases, the weight of the location resolution arc will

become much smaller compared to the normal DNS process.

Definition 4.15: Network Connection Arc

An arc in a WRDG is said to be a network connection arc $a_{c(k,i)}$ if it connects a location resolution vertex $\upsilon_{l(k,i)}$ to a network connection vertex $\upsilon_{c(k,i)}$:

$$a_{c(k,i)} = \langle \upsilon_{l(k,i)} , \upsilon_{c(k,i)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page and $0 \leqslant i \leqslant o$. The weight of this arc denotes the time spend in establishing the network connection between a client and a web server for the object request ***Req(Obj(k,i))*** after the location of the server has been resolved.

In the cases that "persistent-connection" or "pre-connection" is used, the weight of the network connection arc may become much smaller, up to zero.

Definition 4.16: Request Sending Arc

An arc in a WRDG is said to be a request sending arc $a_{s(k,i)}$ if it connects a network connection vertex $\upsilon_{c(k,i)}$ to a request sending vertex $\upsilon_{s(k,i)}$:

$$a_{s(k,i)} = \langle \upsilon_{c(k,i)} , \upsilon_{s(k,i)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page and $0 \leqslant i \leqslant o$. The weight of this arc denotes the time spend in sending out the request message of the object request ***Req(Obj(k,i))*** to a web server through a already-established network connection.

This arc is a critical arc in a WRDG. Its importance is like a "bridge" in a connected graph. This arc can never be removed from a WRDG by any means of transformation. Neither can its weight be reduced to zero.

Definition 4.17: Reply Arc

An arc in a WRDG is said to be a reply arc $a_{r(k,i)}$ if it connects a request sending vertex $\upsilon_{s(k,i)}$ to the first data chunk vertex $\upsilon_{d(k,i,0)}$:

$$a_{r(k,i)} = \langle \upsilon_{s(k,i)} , \upsilon_{d(k,i,0)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page and $0 \leqslant i \leqslant o$. The weight of this arc denotes the time spend by a client in waiting for the first response data chunk being returned from the server after the request message has been sent out. Note that this first returned data chunk mainly contains response headers information of the requested object *Obj(k,i)*, although it may also contain some data of the object body as well.

As there is always at least one chunk of data returning from the web server to the client for every request, this arc always exists. This arc is also a critical arc in WRDG. Its importance is like a "bridge" in a connected graph. This arc can never be removed from a WRDG by any means of transformation. Neither can its weight be reduced to zero.

Definition 4.18: Data Chunk Arc

An arc in a WRDG is said to be a data chunk arc $a_{d(k,i,j)}$ if it connects two successive data chunk vertices $\upsilon_{d(k,i,j-1)}$ and $\upsilon_{d(k,i,j)}$:

$$a_{d(k,i,j)} = \langle \upsilon_{d(k,i,j-1)} , \upsilon_{d(k,i,j)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page and $0 \leqslant i \leqslant o$, j refers to a data chunk in the data chunk sequence that belongs to the object *Obj(k,i)* and $1 \leqslant j \leqslant c$. The weight of this arc denotes the time spend in transferring a successive data chunk from a web server to a client for the requested object *Obj(k,i)*. Note that these successive data chunks usually contain only the body data of the requested object.

There are usually multiple data chunks in the transfer of one object. The content of these data chunks will be interpreted by the client's browser. The order of the interpretation of these data chunks must be in a strict successive sequence order. This

order is represented by the data chunk arcs, i.e. the data chunk represented by the source vertex of a data chunk arcs $a_{d(k,i,j)}$ should always be interpreted prior to the data chunk represented by the target vertex. However, the order of the delivery of these data chunks may be in any order.

Definition 4.19: Ending Arc

An arc in a WRDG is said to be an ending arc $a_{e(k,i)}$ if it connects the last data chunk vertex $\upsilon_{d(k,i,c)}$ to an ending vertex $\upsilon_{e(k,i)}$:

$$a_{e(k,i)} = \langle \upsilon_{d(k,i,c)} , \upsilon_{e(k,i)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page and $0 \leqslant i \leqslant o$, c refers to the last data chunk in the data chunks sequence belonging to the object $Obj(k,i)$. The weight of this arc denotes the time spend in releasing the resources that are occupied by the retrieval process for the object $Obj(k,i)$. Such resources include network connection, CPU, memory space and so on. For the network connection, the released connection is often closed, or handed over to a pool to keep alive for future use (when "persistent-connection" mechanism is adopted).

Definition 4.20: Interruption Arc

An arc in a WRDG is said to be an interruption arc $a_{i(k,i)}$ if it connects from any vertex υ other than the last data chunk vertex $\upsilon_{d(k,i,c)}$ to the ending vertex $\upsilon_{e(k,i)}$:

$$a_{i(k,i)} = \langle \upsilon , \upsilon_{e(k,i)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of the page and $0 \leqslant i \leqslant o$, c refers to the last data chunk in the data chunks sequence belonging to the object $Obj(k,i)$. The weight of this arc denotes the time spend in releasing resources in the case of a premature finishing of the retrieval process for the object request $Req(Obj(k,i))$.

Definition 4.21: Object Deriving Arc

An arc in a WRDG is said to be an object deriving arc $a_{o(k,i)}$ if it connects a data chunk vertex $\upsilon_{d(k,0,j)}$ of the container object of a page ***Page(k)*** to a request initiation vertex $\upsilon_{r(k,i)}$ of an embedded object of the same page:

$$a_{o(k,i)} = \langle \upsilon_{d(k,0,j)} , \upsilon_{r(k,i)} \rangle$$

where k refers to a web page and $0 \leqslant k \leqslant p$, i refers to an object of ***Page(k)*** and $1 \leqslant i \leqslant o$, j refers to a data chunks sequence that belongs to the object ***Obj(k,i)*** and $0 \leqslant j \leqslant c$. The weight of this arc denotes the time spent by the request for object ***Obj(k,i)*** in waiting for its turn to get processed.

The URLs of embedded objects of a page are defined in the container object of the page. As the data chunks of the container object being transferred from web server to client, the content of the data chunks will be interpreted by user's web browser. If there are URLs of embedded objects found in the data chunk being interpreted, new requests for the embedded objects will then be triggered. In other words, the retrieval processes for embedded objects depends on the retrieval and interpretation of the container object. This dependency is captured by the object deriving arcs in a WRDG graph.

Due to certain resource constraints, e.g. limited parallelism width for concurrent fetching of objects, the new request for an embedded object may need to wait for some time before it can get processed. This latency is captured by the weight of the object deriving arc in a WRDG graph.

Definition 4.22: Page Deriving Arc

An arc in a WRDG is said to be a page deriving arc $a_{p(y)}$ if it connects the ending vertex $\upsilon_{e(x,0)}$ of the container object of a page ***Page(x)*** to the request initiation vertex $\upsilon_{r(y,0)}$ of the container object of a page ***Page(y)*** :

$$a_{e(y)} = \langle \upsilon_{e(x,0)} , \upsilon_{r(y,0)} \rangle$$

where x and y refer to two different web pages and $0 \leqslant x \neq y \leqslant p$.

The weight of this arc denotes the time interval from the completion of retrieval of page *Page(x)* to the starting of request of page *Page(y).* This time may include the user's view time on *Page(x)* , the idle time between the two page visits, and the time spent by the request for page *Page(y)* in waiting for its turn to be processed. This waiting time exists mainly due to certain resource limitation, such as CPU time and network connection etc.

In the container object of a page, there are usually hyperlinks pointing to other web pages. When a hyperlink is clicked by a user, the request for the new page will be initiated. In this case, the retrieval process of the next page depends on the retrieval of the current page. This dependency is captured by page deriving arcs in a WRDG graph.

There are also cases that the retrieval of the next page is triggered by user's entering a new URL address in the address bar of his/her web browser. When the retrieval of the new page starts, the retrieval process for the current should be will be either completed already or interrupted prematurely. Here the relationship between the retrieval of the two pages may not be so strong (but there may still be some dependency, for example, the new URL address entered by a user may be hinted by some content of the current page). However, there is still an ordering or precedence relationship between the two pages. The page deriving arc is still used in such situations to capture this ordering or precedence relationship. Here, the page deriving arc may be referred to as Weak Page Deriving Arc. In contrast, the page deriving arc which represents the dependency described in the above paragraph is referred to as Strong Page Deriving Arc.

Definition 4.23: In-Degree of a Vertex in WRDG

The in-degree of a vertex υ in a WRDG graph represents the number operations

that should finish prior to the operation represented by the vertex υ. The operation represented by the vertex υ can start execution when any of its precedent operations completes.

Definition 4.24: Out-Degree of a Vertex in WRDG

The out-degree of a vertex υ in a WRDG graph represents the number of operations that are dependent on the operation represented by the vertex υ . The completion of the operation represented by the vertex υ will cause some or all of its dependant operations to take place.

Definition 4.25: Critical Region of WRDG

The critical region of a WRDG is a sub-graph of it : $G_{cr} = (V_{cr}, E_{cr})$, where V_{cr} and E_{cr} have the following members respectively:

- $V_{cr} = \bigcup_{j=0..c} \{ \upsilon_{d(k,0,j)} \}$
- $E_{cr} = \bigcup_{j=1..c} \{ \langle \upsilon_{d(k,0,j-1)} , \upsilon_{d(k,0,j)} \rangle \}$

Here *j* refers to a data chunk in the data chunk sequence of an object.

The critical region of a WRDG has the following properties:

- Number of vertices and arcs in this region is not fixed.
- This region contains all the source vertices of object deriving arcs. The number of vertices that have object deriving arcs is not fixed. The number of object deriving arcs that a vertex in this region can have is also not fixed (it can be zero).
- The order of the interpretation of the data chunks represented by the vertices in this region is strictly in a successive sequence order. However, the order of these data chunks being transferred may be in any order.

4.3 Three Levels of WRDG

With the understanding of the definitions of the vertices and arcs in WRDM, the WRDG graph can be constructed by carefully monitoring the timing and triggering action of the operations in the retrieval processes for web pages and objects. WRDG graphs can be applied to capture three levels of the web retrieval. These levels are:

1) Intra-Object level WRDG graph
2) Object-level WRDG graph
3) Page-level WRDG graph

Note that WRDG graphs at all levels are to capture the dynamic retrieval processes of web objects and pages, not the static relationship among them. Below we will give the detailed definition and examples of the WRDG graphs at the three levels.

4.3.1 Intra-object level WRDG graph

The WRDG graph at intra-object level is used to capture the retrieval operations sequence of a single object. Therefore, this level may also be referred to as operation-level. The principle of the construction of an intra-object level WRDG graph is straightforward: timing all the operations represented by the vertices in WRDM during the retrieval processes of an object; put an arc between every two successive operations and mark the weight of the arc as the time spent in completing the second operation.

Figure 4.1 gives two examples of intra-object level WRDG graphs. Figure 4.1 (a) shows the retrieval process of an object. In the graph, the vertices represents six operations involved in the retrieval process: initiation of the object request, location resolution for URL in the request, setting up network connection, sending request message to web server, transfer of four data chunks from server to client, and finally, release of the occupied resources such as network connection and memory space etc.

The weights on the arcs are the times required for finishing the operations represented by the vertices to which the arcs are connected. Figure 4.1 (b) shows a similar process of another object's retrieval. But in this graph, the retrieval process is interrupted prematurely after the second data chunk returned from the server. Thus the process goes directly to the ending operation which releases the resources occupied by the request.

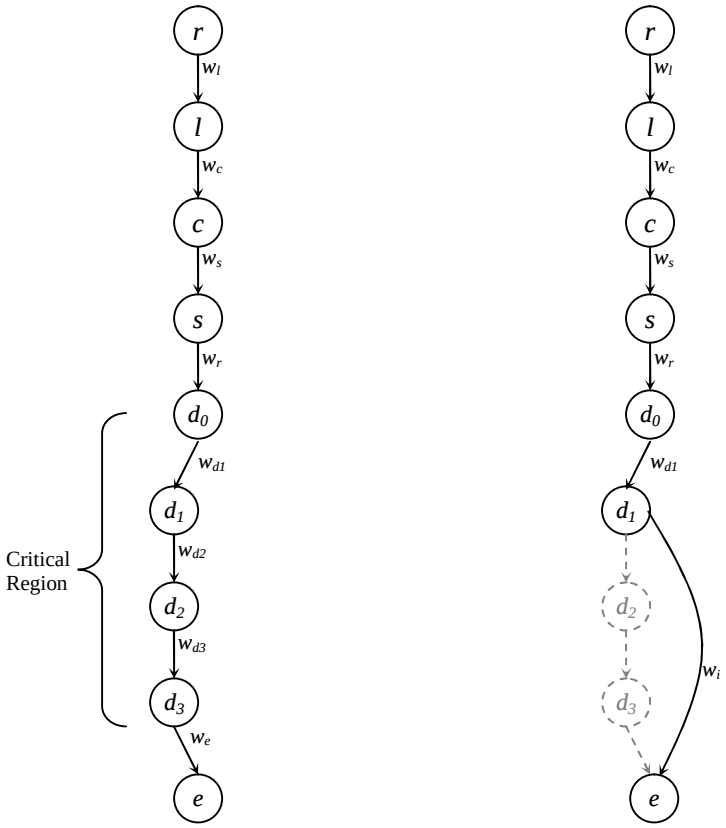


Figure 4.1 Intra-Object level WRDG graph

Note that the intra-object level WRDG graphs can actually include more or less operations than those defined in the previous section. For example, the operation of "access control" can be included in the graph. On the other hand, all the data chunks vertices may be merged into one and treated as the operation of "transfer of object". This change is very flexible and users of WRDM may make their decision based on their needs. For our studies in this thesis, we will stick to the operations represented by

the six types of vertices defined in the previous section.

4.3.2 Object-level WRDG graph

The WRDG graph at object-level is used to capture the retrieval processes of all objects belonging to a page. So, the object-level is also referred to as Inter-Object level or Intra-Page level sometimes. The construction of an object-level WRDG graph can be achieved by constructing an intra-object level graph for every unique object in the page and then connecting them together using object deriving arcs. The starting points of the object deriving arcs can be identified by carefully monitoring the triggering actions between the requests for EOs and the data chunks of the CO. The weight of an object deriving arc is the latency between the time when the data chunk containing the URL of an EO is returned in the CO's retrieval process and the time when the request for the EO is sent out. This latency is the time spent by the request for the EO in waiting for its turn to get processed.

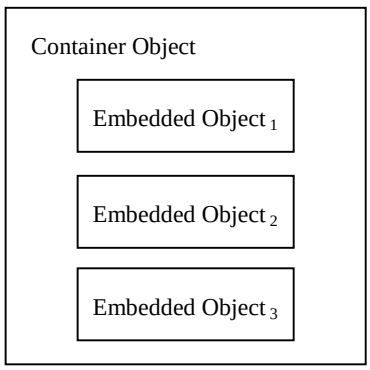


Figure 4.2 A sample web page with three embedded objects

Note that only unique EOs in a web page would appear in the object-level WRDG graph, and each of them should appear only once. The reason for this rule is because that any unique EO in a web page will be retrieved only once even if it is used for multiple times in the page, as the subsequent uses of it will not create any new request since the object will be already made available locally by the first use of it.

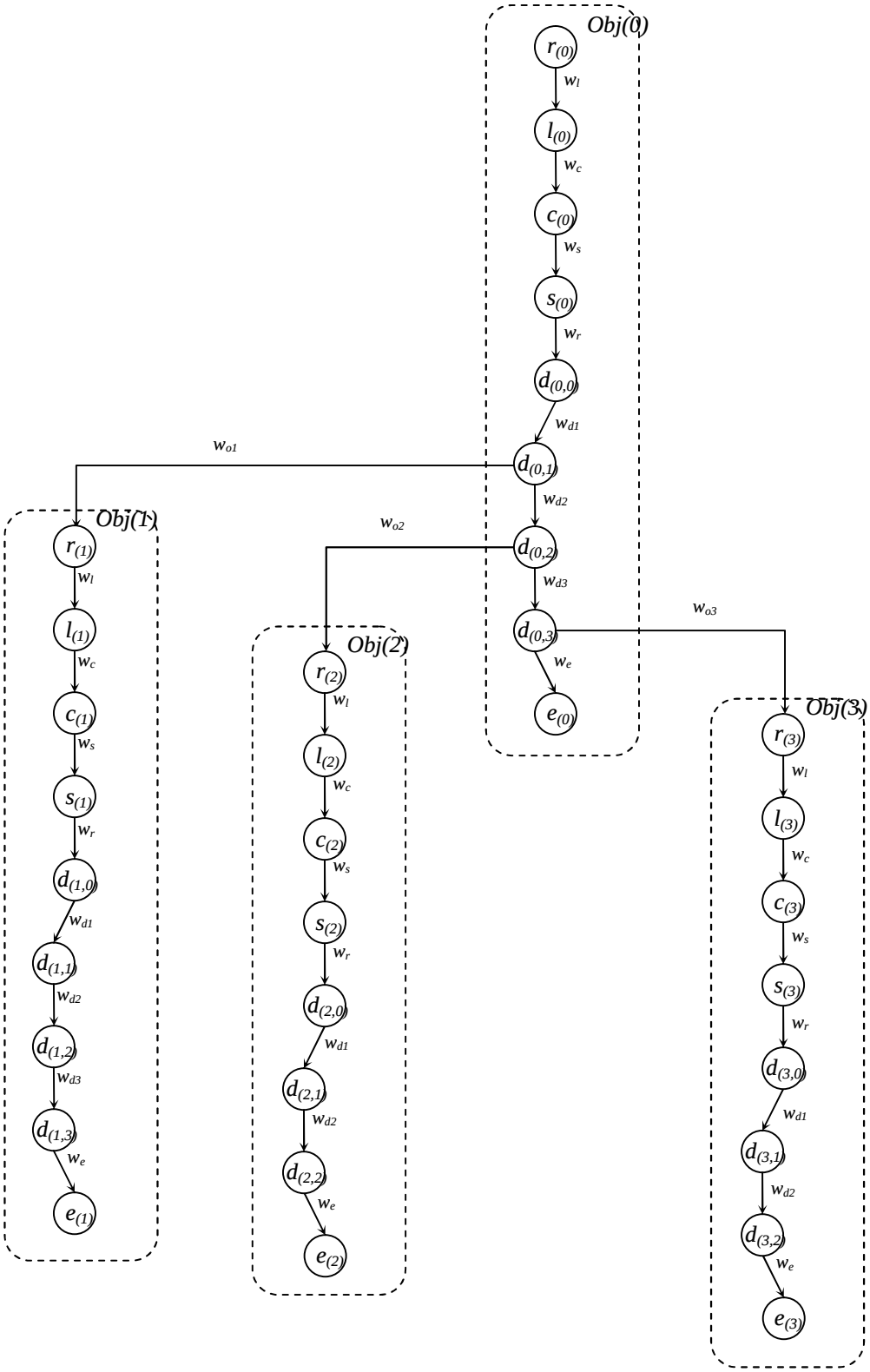


Figure 4.3 Object-level WRDG graph for the retrieval of the page in Figure 4.2

To give an example, suppose we retrieve a web page with three EOs, as depicted in Figure 4.2. The retrieval process for this page may be mapped into the object-level

WRDG graph shown in Figure 4.3. Here we assume the definitions of the three EOs appear in three different data chunk in the retrieval process of the CO.

In the situation when the operation-level details are not important, we may simplify the graph in Figure 4.3 by removing all the details encircled by the rounded rectangle. This would transform the object-level WRDG graph to a simpler graph, as shown in Figure 4.4.

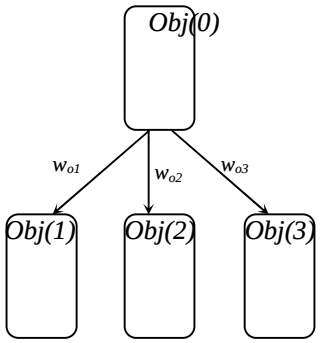


Figure 4.4 Simplified Object-level WRDG graph for the page in Figure 4.2

In the simplified object-level WRDG graph as shown in Figure 4.4, each rounded rectangle represents an object, and it will be referred to as Object Vertex in the rest part of this thesis. The arcs connecting these object vertices are the object deriving arcs. Note that every object vertex actually contains a complete intra-object level WRDG graph which corresponds to the retrieval process of an object.

Theorem 4.1: Single value of in-degree of object vertices

Given an object vertex in a simplified object-level WRDG graph, there is exactly one incoming arc to the vertex. There is one exception which is the object vertex corresponding to the CO, where the in-degree is zero.

Proof: This theorem is based on HTTP protocol and web browsers' behavior on HTML web pages. The retrieval of a web page always starts with the CO of the page. When the content of the CO reaches the web browser, it will be parsed and further requests for the EOs defined in it will be triggered. Once a request for an EO is sent

out, all subsequent accesses to that object within the same page will be served locally. There will not be duplicate transfer between web server and client for the same EO within the same page. □

Due to the single value of in-degree of object vertices, a simplified object-level WRDG graph is always a two-level directed tree with the object vertex corresponding to the CO as the root.

4.3.3 Page-level WRDG graph

The Page-level is also referred to as Inter-Page level sometimes. The WRDG graph at this level is used to capture the retrieval processes of multiple pages which are accessed within a given period of time. A page-level WRDG graph is made up of multiple object-level WRDG graphs. So, the construction of a page-level WRDG graph can be attained by constructing an object-level graph for every page and then connecting them together using page deriving arcs according to the access orders on them. The weight of a page deriving arc represents the time elapsed between two page visits. This time may include the user's view time on the previous page, some possible idle time between the two page visits, and the time spent by the request for the next page in waiting for its turn to be processed. This waiting time exists mainly due to certain resource limitation, such as CPU time and network connection etc.

Note that every unique page should appear only once in the page-level WRDG graph. In the case where a page is visited multiple times within the given period, the number of page deriving arcs pointing to the page will be greater than one.

As we stated earlier, WRDG graphs represent the dynamic retrieval processes of web pages and objects, not the static relationship among them. So, the page deriving arcs may not follow the hyperlinks between pages. For example, there could be cases where two pages are connected by a page deriving arc, but there is no hyperlink in a

page pointing to the other. The situations like this have been discussed in Section 4.2 where we introduced strong page deriving arcs and week page deriving arcs to characterize them.

An example of page-level WRDG graph is given in Figure 4.5. In the example, we assume that there are three successively retrieved pages. Due to the space limitation, we used object vertices in the graph.

Similar to the simplified object-level WRDG graph, page-level WRDG graphs also have simplified version. When situation permits the details to be ignored, page-level WRDG graphs can be simplified by substituting each page with a rectangle and discarding all the object-level and operation-level details. Figure 4.6 shows the simplified version of the page-level WRDG graph shown in Figure 4.5.

In simplified page-level WRDG graphs, each rectangle represents a web page and is referred to as Page Vertex. The arcs connecting these page vertices are the page deriving arcs.

Note that there is no “single value of in-degree” for page vertices. Because a page may be visited for multiple times within a given period, so it may have more than one page deriving arc pointing to it. Due to this property, the page-level WRDG graphs may not be trees.

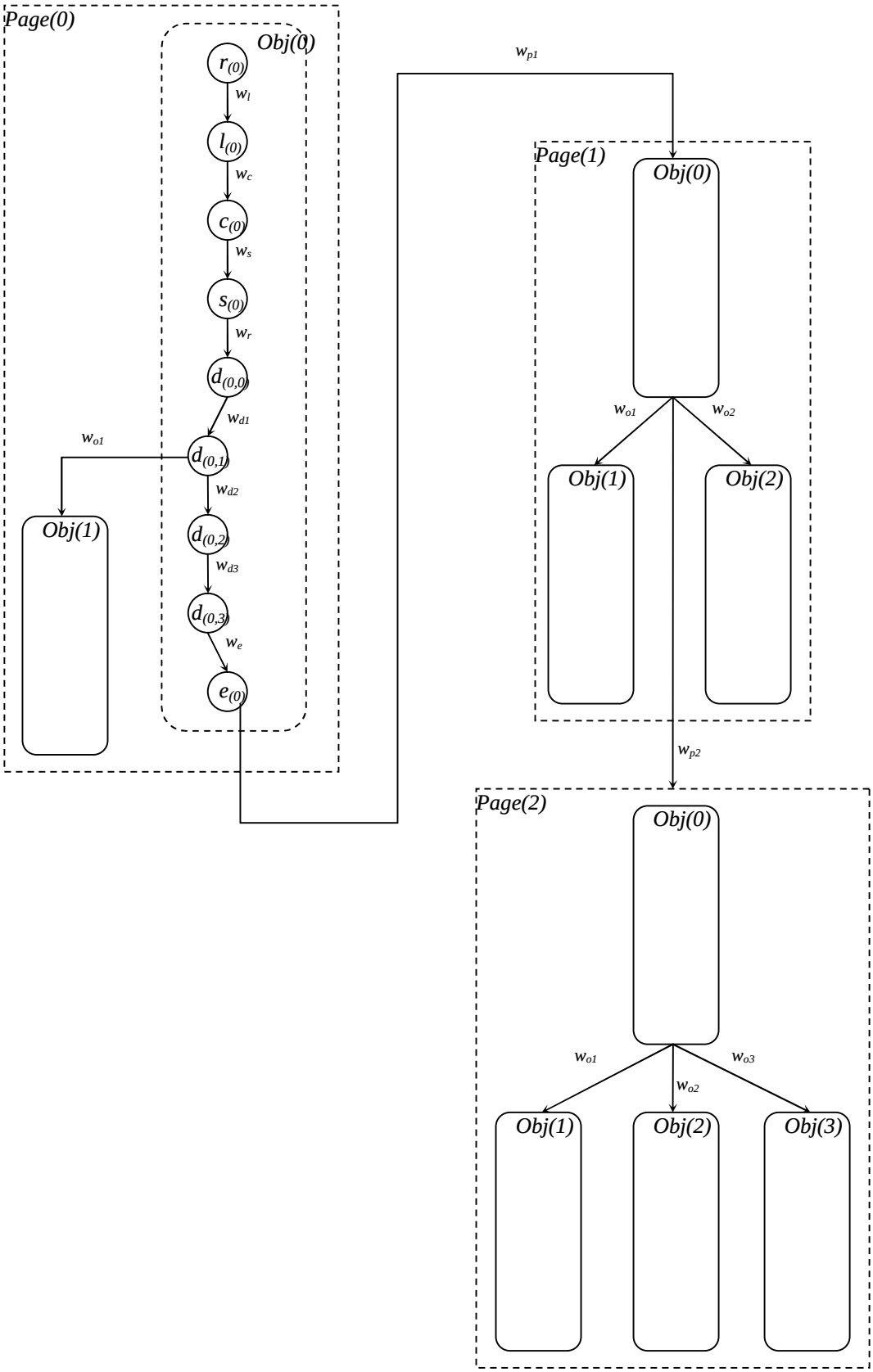


Figure 4.5 Page-level WRDG graph for three successively retrieved pages

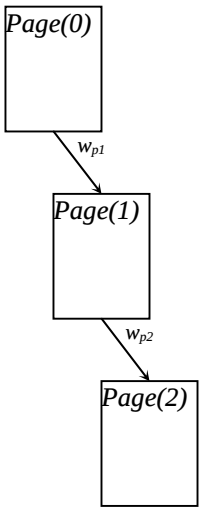


Figure 4.6 Simplified page-level WRDG graph for the graph in Figure 4.5

4.4 Transformation on WRDG graphs

WRDG graphs represent the retrieval process of web objects and pages. Certain transformation can be performed on the standard WRDG graphs. Since WRDG graphs capture the relationship among operations and among objects and reflect the retrieval latency of objects and pages, different transformations on the WRDG graphs will denote different changes to the relationship and the latency of the retrieval processes. The rule of thumb for transformation on WRDG graphs is that the valid web retrieval semantics should be maintained after transformation.

Basically, we can perform the following transformations on WRDG graphs:

Within an Object

1) Changing the weights of arcs

   The weight of an arc stand for the time spent in completing the operation represented by the target vertex. A change to the weight of an arc could mean the completion time of the operation represented by the target vertex is affected by certain mechanism. For example, when DNS caching is used, the weight of the Location Resolution Arc will become smaller. The weights of other arcs may also be changed, which could reflect the impact of some mechanisms like web caching,

prefetching, CDN, persistent connection etc.

2) Removing arcs and vertices

The removal of arcs and vertices always happens in pair, i.e. the removal of an arc would mean the removal of an corresponding vertex at the same time, and vice versa.

When the weight of an arc is reduced to zero, the arc can actually be removed from the graph. Since the weight represents the time for finishing the target operation, it would mean that the target operation takes zero time to finish in this case. Therefore, the vertex representing the target operation should also be removed from the graph. For example, in the case of ideal persistent connection, the weight of the Network Connection Arc may be reduced to zero. In this case, both the Network Connection Arc and the Network Connection Vertex can be removed from the graph.

On the other hand, when a vertex is removed, the arc pointing to it should be removed also. Here is one example for this situation: suppose an encoding mechanism is employed and the number of data chunks in an object transfer is reduced by half, then half of the Data Chunk Vertices will be removed from the object WRDG graph. Consequently, those Data Chunk Arcs connecting to those removed vertices should be removed, too.

When a vertex υ (as well as the arcs connecting to it) is removed, the arcs which are outgoing from υ should be put to connect from the precedent vertex of the vertex υ. Note that within an object WRDG graph, the vertices υ_r and υ_e can not be removed.

3) Splitting one object graph into multiple sub-object graphs

With the support of partial object, an object retrieval may be carried out as multiple

partial requests, each requesting part of the object. Each of such partial requests is a full-fledged HTTP request, only that the number of data chunks of such requests is smaller than the original request. In WRDG graphs, we can capture this situation by splitting the object graph into multiple sub-object graphs, with each of the sub-object graph containing smaller number of data chunks (i.e. smaller Critical Regions). Among the sub-object graphs, there is one called primary sub-object graph, from which other sub-object graphs are derived. So, the vertex v_r of other sub-object graphs should be connected to a vertex υ of the primary sub-object graph. Based on when and where those other sub-object graphs are derived, this vertex υ may vary. But in general, this υ is usually the first Data Chunk Vertex of the primary sub-object graph because that vertex usually contains important HTTP headers (such as "Content-Length") which are essential for carrying out this partial object retrieval.

Between Objects

1) Changing the weights of object deriving arcs

   The weight of the object deriving arc denotes the time spent by the derived object request in waiting for its turn to get processed. This time is mainly affected by the parallelism width for concurrent fetching of objects. So, a change to this weight would generally mean that the parallelism width is changed.

2) Changing the origin points of object deriving arcs

   The origin point of an object deriving arc denotes the place where the object is defined and the request is triggered. Shifting the origin point of an object deriving arc would mean that the definition place of the object has been changed. For example, suppose we have a special mechanism which makes all the embedded objects of a page defined in the first data chunk of the container object, then the

origin points of the object deriving arcs would all be shifted to the first Data Chunk Vertex.

3) Merging multiple object graphs into one

There are some mechanisms which merges the retrieval processes of multiple objects into one process. Existing examples of such mechanisms include pipelining and bundling etc. To capture this situation, the multiple object graphs corresponding to the merged objects can be merged into one graph. In the merged graph, there can be multiple υ_r vertices or only one υ_r vertex depending on the scenario of the mechanisms. But for the vertex υ_e , there is usually only one of it.

By applying different transformation on WRDG graphs, we can map most acceleration mechanisms into WRDG graphs. Because web retrieval latency can be reflected by the weights of the arcs in WRDG graphs, so we can better understand and study the effect of existing and new acceleration mechanisms by investigating their transformation on the graphs.

4.5 Conclusion

In this chapter, we have proposed an operation-level web retrieval dependence model to capture the complex relationship affecting web retrieval latency. This model helps us to understand and study how the retrieval latency of individual objects in a web page contributes to the final page retrieval latency. By constructing WRDG graphs, we clearly see the retrieval dependency between EOs and the data chunks of CO. By taking this into consideration, we would be able to compute more precise page latencies than those object-level based studies. Also, the effects of different web retrieval mechanisms can be illustrated by different transformation on WRDG graphs.

It is worth mentioning that our model is different from two research works which also proposed graph models for web study. One work is from IBM [277] which

proposed a simple object dependency graph to describe web page structure. However, its objective is to describe the dynamic nature of web data and is mainly used for the synchronization of database objects, not for the study of web page retrieval latency. Another work also proposed graph for the study of web [278, 279, 280, 281, 282, 283]. However, the graphs in these studies capture the static structure of web page, with web pages represented by nodes and hyperlinks represented by directed edges. Such graphs are used for algorithms such as ranking pages and finding natural communities of pages etc. Our WRDG graphs capture the retrieval process of web pages, which are used for the study of web retrieval latency. Furthermore, our WRDM also works at operation and chunk level, which is not achieved in other studies.

Chapter 5 Experimental Environment and Tools

Our studies on web content delivery are based on comprehensive detailed trace-driven simulation experiments and real system testing. In this chapter, we would like to describe the experimental environment and tools used in our studies.

5.1 Web Access Model

In our studies, we always assume the use of a proxy server as an intermediary between the client and the web server. That is, we assume the following web access model shown in Figure 5.1.

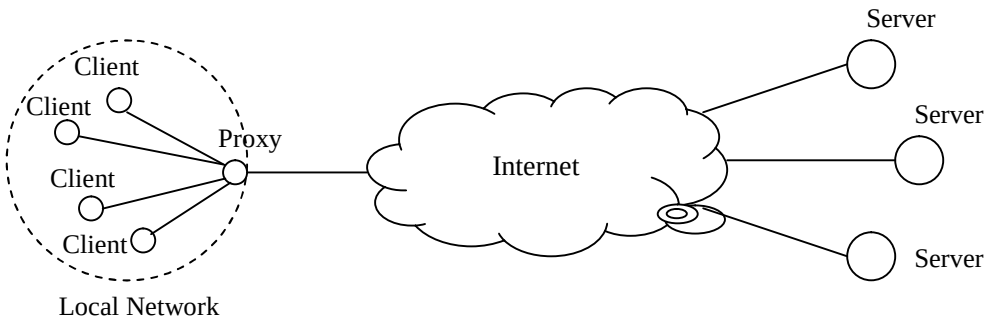


Figure 5.1 Web access model

There are two reasons why we always use a proxy in the system. Firstly, it is because this configuration is very common in the web today [13]. Secondly, we use proxy to perform two important tasks:

1) Recording logs for web retrieval

Because a proxy server can monitor every detail of web retrieval processes, so we make it to record very detailed information about them. The information recorded includes timing measurements of object retrieval at fine-granularity level, header information of HTTP requests and responses, cacheability information, chunk-level information about the retrieval, and many others.

When situation permits, we assume the information logged by the proxy is the information seen by the clients, e.g. the timing measurements. It is equal to say that

the whole dashed-line circle in Figure 5.1 is treated as one client in some situations. This assumption is reasonable because the proxy server is in the same LAN where the clients exist. In fact, the clients and the proxy are connected by direct physical network connections. So, the latency between proxy and clients is negligible compared with the latency between proxy and remote web servers. We expect this assumption does not affect the correctness of our results.

2) Implementing and testing our proposed new mechanisms.

A proxy server has full control on the web requests passing through it. With this capability, we will be able to instrument the proxy to implement our new mechanisms, for example, issuing requests at earlier stage based on certain knowledge. It would be very difficult (if not impossible) to carry out such work on most common web clients like MS-IE and Netscape.

Some experiments may require detailed retrieval information on servers, or, some mechanisms need to be implemented on servers (such as compression). In these situations, we will put a reverse proxy in front of the server and make the proxy to perform the tasks for the server. Here, we will treat the union of reverse proxy and server as a whole web server. Figure 5.2 illustrates this situation. (in contrast, the proxy used in front of clients is called forward proxy.)

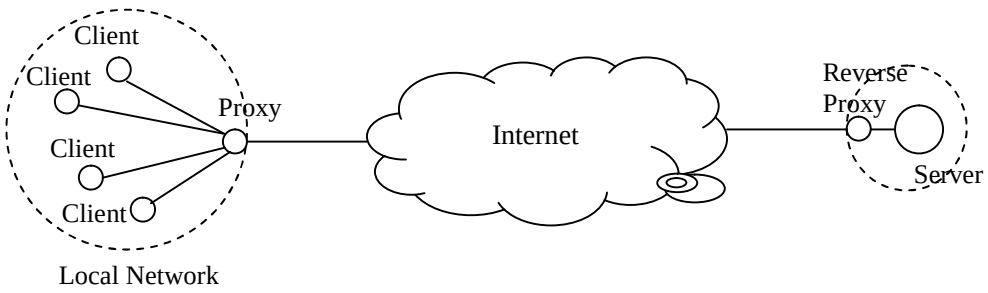


Figure 5.2 Web access with reverse proxy

In Figure 5.2, the union of the reverse proxy and the server should be physically far away from the clients in order for the experimental results to reflect the effect of

real web system. However, to physically place a machine considerably far away is not often achievable. So we use another way to emulate this situation. The idea is to employ a remote proxy server. Figure 5.3 shows the structure of this system.

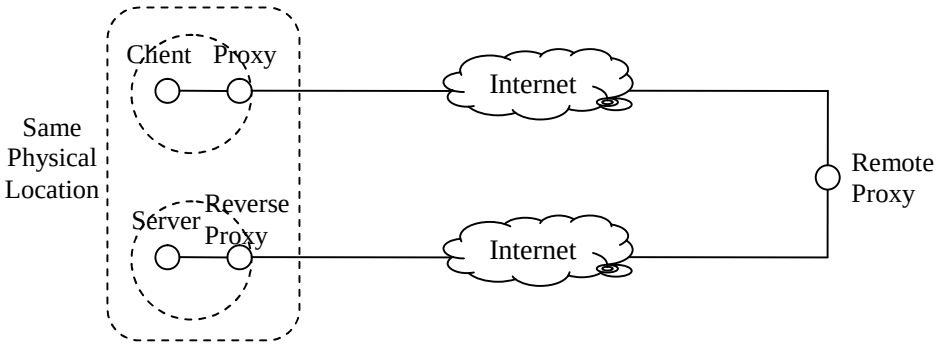


Figure 5.3 Web access with remote proxy

In the above system, both the client and the server can be placed physically in the same location. But the traffic between them will be directed to a remote proxy in the Internet. For the remote proxy, we purposely chose those ones physically located in other continents, e.g. those in Europe and America[1]. Therefore, the experimental results such as retrieval latencies would be comparable to the results got from real web systems.

5.2 Experimental Tools

Our experiments involve both real software tools of web client, proxy, and server etc. and simulators. To facilitate system implementation, most of the software tools we used are open source systems. Below we give a brief description of the tools used in our study.

The web client programs we used are Wget [284], Pavuk [285], MS-IE [70], Netscape [71], and some simple client programs written by ourselves. Wget and Pavuk are free utilities for non-interactive retrieval of files from the web. Because they are non-interactive command-line tools, so it is easy to call them from scripts. Therefore,

[1] Examples of the remote proxies we used: cache.bt.net:3128, webcache.bt.net:8080, 80.49.22.130:8080, 80.18.158.154:8080

they are ideal for running large scale experiments since the whole process can be controlled by scripts automatically. For the experiments which require small number of runs, the interactive clients MS-IE and Netscape may be used. In some cases, we also write our own client programs for special purposes, for example, the client that retrieves only HTTP headers for given URLs. The versions of the client programs we used are as follows: Wget 1.4.5, Pavuk 0.9p128, MS-IE 5.5, Netscape 4.7 .

Proxy server is generally used to retrieve web objects on behalf of clients and collect statistics about the traffic on the network. We choose to use Squid [73] as the proxy server in our experimental environment because not only it is the most popular web proxy server in use today, but also it is a free, open-source software which enables us to instrument it to collect special information that we need, and to implement and test our new mechanisms based on it. The version we used is Squid 2.4.STABLE3, which was the latest version as at the time of our study. For the reverse proxy, we also used Squid and the version is the same.

The web server program we choose is also a free, open-source software, which is Apache, version 2.0.39. Apache has been the most widely used web server on the Internet since 1996 [72]. By using it as the web server in our experimental environment, we would expect that the results we have got would be realistic and applicable to most real web servers on the Internet.

For those experiments which retrieve web pages directly from the Internet, the web servers of real websites on the Internet are involved. These would include all kinds of web servers found in the current web system, such as Apache, MS-IIS, NCSA HTTPd etc.

Besides the above software tools from public domain, we also wrote our own simulators for our experiments. While many results in our study are obtained from real

system experiments using the above software tools, there are situations where it is difficult (or even impossible) to get results from real systems experiments, for example, varying the parallelism width for retrieval. For those situations, we write simulators and run trace-driven simulations to get the results. Because the traces are got from real web systems, so we expect the results from such simulations can reflect the real situations quite accurately.

5.3 Software/Hardware Platform and Network Environment

The operating system we used is Red Hat Linux release 7.2 (Enigma), Kernel 2.4.7-10 on an i686. The compiler used to compile the software tools is gcc version 2.96. For scripts written in shellscript or perl, the interpreters are GNU bash version 2.05.8(1)-release (i386-redhat-linux-gnu) and perl v5.6.0 built for i386-linux.

The typical configuration of the machines we used is as follows:

Processor	Intel(R) Pentium(R) 4 CPU 2.00GHz
Memory	512Mbytes (with 1GB swap space)
Network Adaptor	3Com Corporation 3c905C-TX (Fast Etherlink) (rev 120)
Hard Disk	20GB--40GB

The connection between client and forward proxy is direct physical connection. The same situation applies to the connection between server and reverse proxy (when reverse proxy is used). For the Internet connection, our environment is the Singapore Advanced Research Network in the National University of Singapore, which has 45 Mbits link to the U.S.. The Internet traffic in our environment is quite heavy and diverse. So we expect that the results we got in such an environment would well represent the situations encountered by typical broad-band Internet users.

5.4 Obtaining Logs

Web logs are valuable data often used by researchers for experiments and comparison for their studies on web content delivery. There are mainly two types of logs, namely proxy logs and server logs. Proxy logs are recorded by proxy servers, and

the data in such logs are often used to represent the measurements seen at client side. Web servers often record the requests they served into server logs. To study the characteristics of the traffic or web pages and objects on a web server, we would need to use the server log. In our study, both proxy logs and server logs are used.

Note that logs are also frequently referred to as traces. In this thesis, I will use these two terms interchangeably without differentiating them.

We obtained proxy logs from two sources. One is the traces from the National Laboratory for Applied Network Research (NLANR) [276]. The other source is the logs generated and collected by our own systems.

1) From NLANR

NLANR traces are the most popular, up-to-date, real proxy traces available to researchers. NLANR's hierarchical proxy system adopts Squid proxy caching software, which is the same software as we used in our study. Their proxy system consists of about ten proxy servers. The traces of all the proxy servers are published on their web site daily. Three NLANR traces have been randomly chosen for our experiments. They are the traces on 12th March 2002, 8th January 2003 and 5th August 2003. Note that we only used one of the ten proxy logs they published on each of the three days. Each of the traces we used contains about 1.2 – 1.4 million requests.

2) From our own system

Some of our experiments require special information about web retrieval which is not available in NLANR traces. For example, the HTTP headers and chunk information are essential to our experiments but they are not provided in NLANR traces. In order to get such information, we built systems to collect it.

We first obtained URL addresses of web objects or pages from NLANR traces. Then we fed them into systems similar to those shown in Figure 5.1 and Figure 5.3 to

replay real retrieval for those URLs and gather logs during the process. We have instrumented the proxy servers to make it record very detailed, chunk-level information for every object retrieval at both client side and server side. Special client programs are also built for the collection of special information, for example, the position of embedded objects in the body of container object of web page.

The collection of special log was performed near the date of the original NLANR traces from which the URL addresses were sampled. This assures that the information we obtained should be very close to the actual values that would have been obtained at the time when the original NLANR traces were logged.

We have logged information for a large number of web objects and pages, which is big enough for generating representative statistics results from them.

The availability of server logs is much smaller than proxy logs. We only managed to get a server log from the website of School of Computing (SoC), National University of Singapore (NUS). The log is dated on 30th October 2000 and contains about 85,000 entries.

Logs often need to be pre-processed before then can be fed into simulators for doing experiments. The pre-process usually carries out the following tasks. First, because logs generally contain a wide range of information and much of the information may not be necessary to the simulations, so we need to extract the useful information. Second, the original format of the logs may not suitable to the requirement of simulators. So we need to convert the format of them. Last, some information may be recorded into multiple log files. It is often necessary to consolidate the information into one file.

5.5 Getting Results

The results presented in our studies are obtained either from trace-driven

simulations or real system testing. We built a wide range of simulators for carrying out comprehensive experiments. Pre-processed logs were fed into the simulators to generate the results.

When situation permits, we also built real systems to do our experiments. For example, the experiments of compression were conducted on real systems. The systems are built based on the tools described in Section 5.2, and the structures of the testing systems are similar to those shown in Figure 5.1 and Figure 5.3. The network environment is stated in Section 5.3.

5.6 Summary

In this chapter, we described the experimental environment and tools used in our studies. By adopting open-source systems, we are able to record very detailed logs and implement our new mechanisms in real systems. The experiments in our studies are carried out based on trace-driven simulations as well as real system testing. The logs we got are huge and comprehensive. They are big enough for generating representative results. The environment of our study is typical in the Internet, so the results we got shall have good representativeness of the situations encountered by most other web users.

Chapter 6 Analysis of Web Retrieval Latency Using WRDM Model

6.1 Introduction

Web retrieval latency has been the focus of study due to the exponential growth of the web. The current web system is made up of pages containing html, image and other types of objects. Many previous studies focus on the retrieval latency of objects. However, this is insufficient and sometimes inaccurate because the unit of web browsing is web page instead of object. To web users, page retrieval latency is more meaningful. While page retrieval latency is derived from object retrieval latency, the mapping between them is not that direct and simple. In order to well understand web retrieval latency, especially page retrieval latency, we shall go into more detailed level of web retrieval.

The transfer of an individual web object is typically delivered in a sequence of data chunks, and the characteristics of chunk sequence transfer have great impact on object retrieval latency. When objects are put together to form pages, the interaction among the objects become very complicated. In a web page, there is a primary object called container object, which contains the definitions of other objects (embedded objects) of the page. Because of this, the retrieval of the embedded objects highly depends on the retrieval process of the container object of the page, and this dependency will incur significant delay to retrieval latency for the embedded objects. Furthermore, current web system employs parallelism for parallel fetching of objects, which makes it possible for the retrieval of some objects to virtually have no effect on the total page latency. All these factors make the mapping from object latency to page latency very complicated, and they are largely ignored in previous object-level studies on web retrieval latency. In this chapter, we would like to research on the web retrieval latency from operation and chunk level based on our WRDM model. By detailed

investigation on the interaction between operations and between objects, it gives us an insight view on the root-cause of web retrieval latency and how those factors are greatly interacted in determining page retrieval latency.

With the exponential growth of web usage and the development of pervasive web content delivery, web content transformation emerges as an important technology to satisfy the different expectation of web users. While there are many studies on web content transformation, their major focuses are on the functionalities and real-time features. There is little study on the possible impacts of content transformation approaches on web retrieval latency. In this chapter, we would like to analyze the performance impacts of content transformation using our WRDM model.

We also derive upper bounds on the performance improvement for acceleration mechanisms in this chapter. While many mechanisms have been proposed and shown promising potential of acceleration, it remains to be seen the quantitative upper bound of them. Based on the understanding, analysis and results of object and page retrieval latency revealed under our model, we derive two upper bounds for acceleration mechanisms, which help us to understand the potentials of web acceleration.

In order to obtain enough information for studying chunk level characteristics and factors affecting page latency, we have re-run about a million requests and recorded very detailed logs, including timing information of operations, chunk data information, content of each page, and the structure of each page, etc. The traces, tools and environment for running these experiments are described in Chapter 5. Some results presented in this chapter are obtained directly from the experiment logs, some others are obtained from simulations based on the detailed logs.

6.2 Analysis of Object Fetch Latency

First of all, we would like to investigate the retrieval latency for objects. We

study is at operation-level and chunk-level, which can give us in-sight view of object latency.

6.2.1 Latency Components of Object Latency

Before we proceed to the detailed analysis of object fetch latency using WRDM, we first give precise definition of the retrieval latency for web objects to clarify possible ambiguity.

Definition 6.1: Object Fetch Latency

The retrieval latency of an object is defined as the time from the initiation of the request for the object, successful transfer of data chunks, until the release of resources that are occupied by the request.

We define the release of resources (like network connection) to be the ending point of object retrieval. This is because that if the resources that are occupied by an object request are not released, the retrieval process for that object would be considered still under processing. A visual evidence of this uncompleted processing to user may be that the progress bar in his/her web browser is still on the move. So user will perceive that the retrieval process is still ongoing. Therefore, the “real completion” of the retrieval of an object should be considered reached only when the occupied resources are released.

The release of occupied resources can occur in two situations. One situation is the natural ending process following the arrival of the last data chunk of the requested object. In this case, all the content of the requested object has been transferred from server to client, and the release of resources follows naturally. The other situation is that the retrieval process for an object is interrupted prematurely. When such situation happens, all the resources that are occupied by the request will be released immediately and the process for the request is considered finished. Among these two

situations, we will use the first one as the default situation in our study, unless otherwise stated. Why we make this choice is because that: in most cases, the full presentation of an object cannot be achieved until all its data has returned from the server. Although this restriction may not be so obvious for progressive objects like JPEG2000 images where the presentation of such objects can be carried out in an accumulative way as their data are made available chunk by chunk, we argue that: (1) progressive objects have not got its popularity on the web yet, the majority of web objects are not progressive objects; (2) even for a progressive object, its presentation cannot be considered as fully completed unless all of its data chunks have returned from server because otherwise a partial content or lower resolution image/file will be experienced by user. So, it is reasonable for us to set the default situation as the one which requires all data chunks of an object to be returned from server and the retrieval process ends naturally.

Put the above two points together: By default, we measure the retrieval latency of an object as the time from the initiation of the request, counting in the transfer time for all the data chunks corresponding to the whole object, till the release of resources that are occupied by the request. The situation of premature interruption of retrieval process will not be taken into consideration unless otherwise stated.

Mapping the definition of object fetch latency into an intra-object WRDG graph, the object fetch latency for an object *Obj(i)* is represented by the distance of the path from the request initiation vertex $\upsilon_{r(i)}$ to the ending vertex $\upsilon_{e(i)}$. In the rest parts of this thesis, we may refer to this path as Object Retrieval Cost Path.

Note that the “distance” of a path refers to the sum of the weights of the arcs along the path. In contrast, the “length” of a path refers to the number of the arcs along the path. We will use these two terms distinctively in our work.

With the understanding about the details of retrieval processes described in WRDM model (see Chapter 5), the retrieval latency for an object can be divided into five components: (1) location resolution time, (2) connection time, (3) request sending time, (4) chunk sequence time, and (5) ending time. Below we give the precise definitions of these five components of object fetch latency.

Definition 6.2: Location Resolution Time (LRT)

Given an object request *Req(Obj(i))*, the location resolution time of the object *Obj(i)* is defined as the time from the time when the request is initiated to the time when the location where the request should be forwarded to is resolved.

In the intra-object level WRDG graph representing the retrieval process for the requested object *Obj(i)*, the location resolution time of the object is given by the weight of the location resolution arc $a_{l(i)}$ connecting the request initiation vertex $\upsilon_{r(i)}$ to the location resolution vertex $\upsilon_{l(i)}$, where $0 \leqslant i \leqslant o$.

Definition 6.3: Connection Time (CT)

Given an object request *Req(Obj(i))*, the connection time of the object *Obj(i)* is defined as the time from the time when the location of the destination server is made known to the time when a network connection between the client and the destination server has been established.

In the intra-object level WRDG graph, the connection time of an object *Obj(i)* is given either by the weight of the network connection arc $a_{c(i)}$, where $0 \leqslant i \leqslant o$.

Definition 6.4: Request Sending Time (RST)

Given an object request *Req(Obj(i))*, the request sending time of the object *Obj(i)* is defined as the time from the time when a network connection between the client and the destination server has been established to the time when the request message has been delivered from the client to the server through the connection.

In the intra-object level WRDG graph representing the retrieval process for the requested object *Obj(i)*, the request sending time of the object is given by the weight of the request sending arc $a_{s(i)}$ connecting the network connection vertex $\upsilon_{c(i)}$ to the request sending vertex $\upsilon_{s(i)}$, where $0 \leqslant i \leqslant o$.

Definition 6.5: Chunk Sequence Time (CST)

Given an object request *Req(Obj(i))*, the chunk sequence time of the object *Obj(i)* is defined as the latency time from the receiving of the first data chunk *Chk(i,0)* of the object to the receiving of the last data chunk *Chk(i,c)* of the object, where $0 \leqslant i \leqslant o$ and $c = g(i)$.

In the intra-object level WRDG graph representing the retrieval process for the requested object *Obj(i)*, the chunk sequence time of the object *Obj(i)* in the page is given by the distance of the path from the request sending vertex $\upsilon_{s(i)}$ to the last data chunk vertex $\upsilon_{d(i,c)}$ corresponding to the transfer of the object.

Definition 6.6: Ending Time (ET)

Given an object request *Req(Obj(i))*, the ending time of the object *Obj(i)* is defined as the latency time from the receiving of the last data chunk *Chk(i,c)* of the object to the release of the resources occupied by the object request, where $0 \leqslant i \leqslant o$.

In the intra-object level WRDG graph representing the retrieval process for the requested object *Obj(i)*, the ending time of the object *Obj(i)* is given by the weight of the ending arc $a_{e(i)}$ connecting the last data chunk vertex $\upsilon_{d(i,c)}$ to the ending vertex $\upsilon_{e(i)}$ of the object.

Figure 6.1 gives an example intra-object level WRDG graph which illustrates the five latency components defined above.

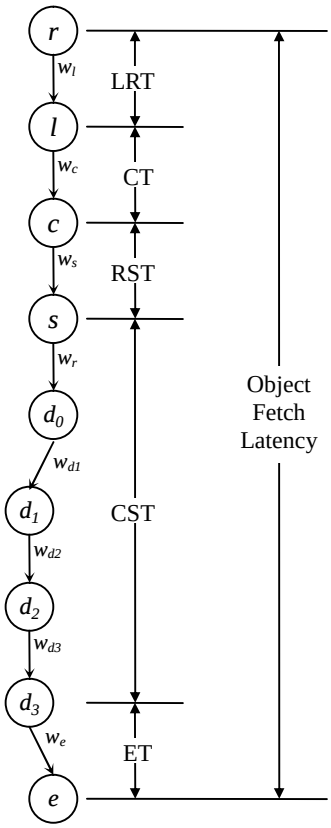


Figure 6.1 Latency components of object fetch latency

The chunk sequence time CST usually consists of the transfer time of multiple data chunks. When we need to refer to the transfer time of a specific data chunk *Chk(j)*, we will use this symbol CST(*Chk(j)*) , e.g. CST(*Chk(0)*) stands for the transfer time of the first chunk *Chk(0)* , CST(*Chk(3)*) stands for the transfer time of the chunk *Chk(3)* , so on and so forth.

Among the latency components, the request sending time RST and the first chunk time CST(*Chk(0)*) are difficult to record in reality because they requires to record timing information at both client side and server side. To deal with this problem, we adopt two compromise ways in our study:

1) Use HTTP-RTT time instead of RST time

While it is difficult to record individual RST time and CST(*Chk(0)*) time, it is easy to record "RST + CST(*Chk(0)*)" at client side. The time of "RST + CST(*Chk(0)*)" is actually the time span from the time when the client starts to send out request to the

time when the first data chunk *Chk(0)* returns from server. This time span is like the round trip time (RTT) in TCP transaction. We refer to this time as the HTTP-RTT time in our study. In the situation where knowing individual RST time and CST(*Chk(0)*) time is not particularly important, we may use this HTTP-RTT time for our study. For example, in the case when we focus our study on the latency component LRT, to further distinguish other latency components may not be useful, then we can just use HTTP-RTT or even a more coarse timing measurement.

Figure 6.2 plots HTTP-RTT time in the object fetch latency.

2) Approximate RST and CST(*Chk(0)*)

In the cases where individual RST time and CST(*Chk(0)*) time are required, we would use the following way to approximate them:

First, we use CST(*Chk(1)*) to approximate CST(*Chk(0)*), i.e. we consider CST(*Chk(0)*) to be the same as CST(*Chk(1)*). This is reasonable because the times of the first two data chunks should be very close. Because CST(*Chk(1)*) is easy to record, so we can get CST(*Chk(0)*) easily. In the case where the object transfer consists of only one chunk (i.e. there is no *Chk(1)*), we can use the statistical value of CST(*Chk(1)*) from other objects' transfer for the approximation.

Second, for RST, we will approximated it as "HTTP-RTT – CST(*Chk(1)*)". As stated above, HTTP-RTT is easy to measure and it is made up of RST and CST(*Chk(0)*), so naturally we can use "HTTP-RTT – CST(*Chk(1)*)" to approximate RST.

Lastly, we would like to point out that we can actually include more latency components in our definitions or divide object latency into more detailed components. For example, the time for performing access control can be included in the definition. On the other hand, the CST(*Chk(0)*) time can be further divided into two (or even more)

sub-components[2]: one is the time spent by server for reading the object from disk into its main memory, the other is the time for the actual transfer of the first data chunk *Chk(0)*. Our model and definitions can be easily extended to cover such situations. However, for our study in this thesis, we would stick to the five latency components defined above for object fetch latency.

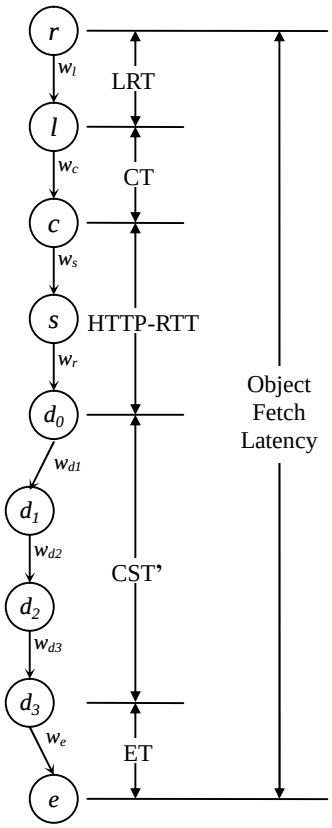


Figure 6.2 HTTP-RTT time in the object fetch latency

6.2.2 Experimental Study and Analysis

Firstly, let us review some statistics about object retrieval at object level, as done in many other works. Figure 6.3 plots the distribution of objects with respect to object sizes. From this graph, we see that the majority of web objects are quite small in size, with an average size of about 5.71 KBytes.

Figure 6.4 shows the distribution of average object fetch latency with respect to object sizes. We see that when objects size is small, e.g. less than 8 Kbytes, the

[2] Because of this reason, the above approximation of RST and CST(*Chk(0)*) may not be so accurate. For studies that require high accuracy of RST and CST(*Chk(0)*), we do not recommend this approximation.

retrieval latency for smaller objects is often comparable to that of bigger objects. For example, the retrieval latency for a 1 KByte object is similar to that of a 4 KBytes object. This could be due to the relatively large time required for setting up network connection. For small objects, their actual transfer time is relatively small compared with the network connection setup time. So, the increase in object size will not affect the whole object latency much. However, for big objects with size larger than 16 KBytes, their retrieval latency does increase with the increase in object size. This is because the network connection setup time becomes relatively small and the actual transfer time becomes the dominating factor of whole object latency for big objects. With bigger size, objects would need longer transfer time. So the retrieval latency would increase with the increase in object size.

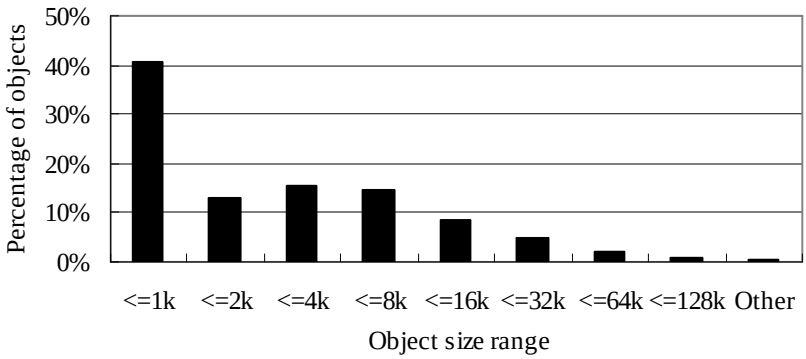


Figure 6.3 Distribution of objects w.r.t. object size

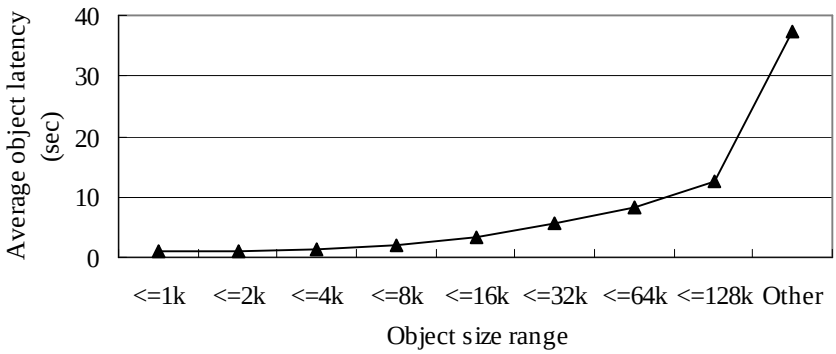


Figure 6.4 Distribution of object latency w.r.t. object size

Next, we would like to investigate the relative distribution of the five latency components in object retrieval, as defined in the previous section. Figure 6.5 plots the

relative distribution of those latency components against the size of the objects. Note that in this graph we used approximated RST and CST(*Chk(0)*) because they are difficult to record in our experiments. Instead, we recorded the HTTP-RTT time and compute approximated RST and CST(*Chk(0)*) based on it, as described earlier on. This approximation will have little impact on the correctness of the overall distribution of those latency components.

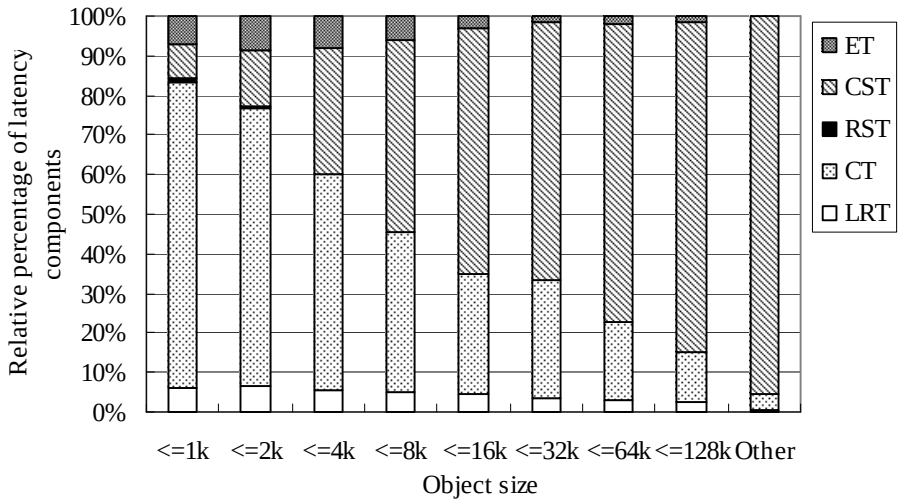


Figure 6.5 Relative distribution of latency components w.r.t. object size

From Figure 6.5, we see that the connection time CT and chunk sequence time CST are the two major latency components in object retrieval. They together take up over 85% of the whole object fetch latency for objects with all sizes.

The connection time CT is generally more significant than CST for objects with smaller sizes. For objects with size less than 4 KBytes, CT time occupies more than 50% of the whole object latency. Considering that the majority of objects have sizes less than 4 KBytes (see Figure 6.3), we expect that CT time plays an important role in object fetch latency. This re-confirms the importance and effectiveness of connectivity-based acceleration mechanisms such as persistent connection, pre-connection, etc. Bundling [23, 24, 25] also helps in reducing CT times by bundling multiple objects into one and using only one network connection for it, which removes

the need of setting up multiple connections for each of the objects.

The relatively large distribution of CT time also explains why the retrieval latency for smaller objects is often comparable to bigger objects for objects under 4 KBytes, as we observed in Figure 6.4. For this group of objects, their retrieval latency is dominated by CT time. The increase in object size only affects the CST time, which is relatively small for objects with sizes smaller than 4 KBytes. So, the retrieval latency for smaller objects is similar to that of bigger objects for this group of objects.

As object size increases, the relative percentage of CT time becomes smaller while CST time gains its significance. For objects with sizes greater than 8 KBytes, the CST time starts to occupy more than 50%, up to about 95% of the whole object latency. This is understandable as large objects generally require longer time for the actual transfer of their content. As there is also a considerably large percentage (about 30%) of objects belonging to this group, this indicates that the acceleration mechanisms which aim to reduce the actual transfer latency would be also effective. Some existing examples of such mechanism include encoding (like compression), transcoding and content selection etc. For very large objects, intra-object parallelism may be used to improve the transfer latency.

Other three latency components, namely LRT, RST and ET, are relatively small. However, they are still not negligible, especially for small objects which define the majority of web objects. The location resolution time LRT comes from the DNS process in current web system. Figure 6.5 shows that LRT time contributes from less than 1% to more than 6% of the whole object latency. This suggests that there is still some room for improvement on top of the current DNS system.

For RST and ET, they are relatively smaller and it is often difficult to reduce them. Because any web request would at least involve the transfer of a request message

from client to server and the release of resources occupied by the request, so these two latency components would always be there and they are mainly determined by hardware infrastructures such as network bandwidth and computing power. Besides upgrading the hardware, some software approaches which may help in reducing these two components include parallel fetching and bundling. While parallel fetching tries to hide RST and ET times by overlapping them with each other, bundling tries to share one RST and ET time among multiple requests so that the average RST and ET time for each single request becomes smaller.

Note that the above observation is valid for individual single object retrieval only. When objects are put together to form pages, some of the observation may not hold any more. For example, when page retrieval latency is mentioned, CT and CST times may not be the largest latency components any more. We will discuss this further shortly later.

Below, we present some chunk-level studies about single object retrieval.

The transfer of an object from server to client typically involves a series of data chunks. Figure 6.6 plots the distribution of objects with respect to the number of chunks in the transfer. From the graph, we see that while more than 40% of objects contain only one data chunk in their transfer, the majority of other objects consist of multiple data chunks. On average, an object has about 6.5 data chunks in its transfer.

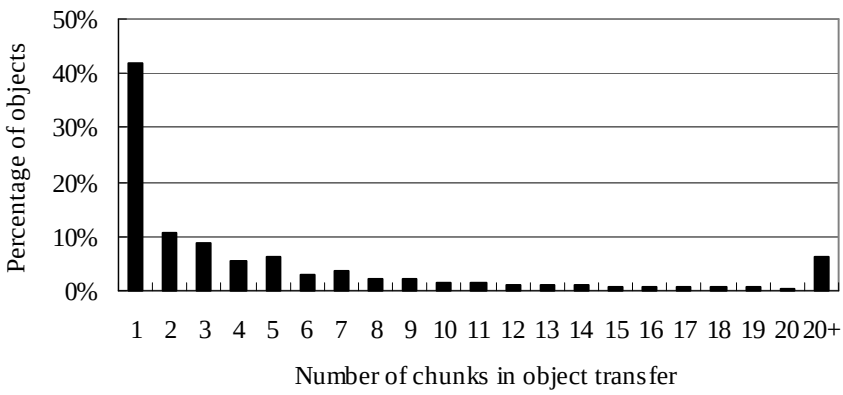


Figure 6.6 Distribution of objects w.r.t. number of chunks

Next, we would like to study some properties about data chunks. Figure 6.7 and Figure 6.8 show the distribution of chunks sizes and latencies respectively. Figure 6.7 plots the distribution of chunks with respect to chunk size. It shows that the majority (65%) of chunks have typical sizes between 1 KBytes and 2 KBytes. However, there are chunks with much bigger sizes, and that puts the average chunk size at 5.3 KBytes. We also note that there are a high percentage of chunks with sizes larger than 10 KBytes.

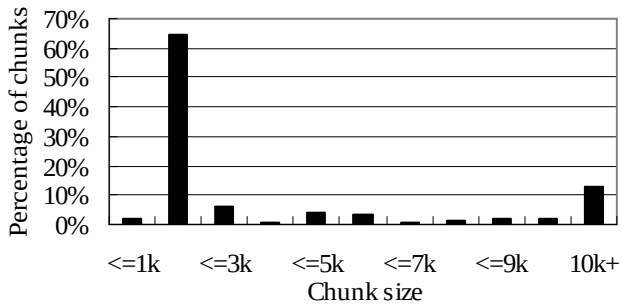


Figure 6.7 Distribution of chunks w.r.t. chunk size

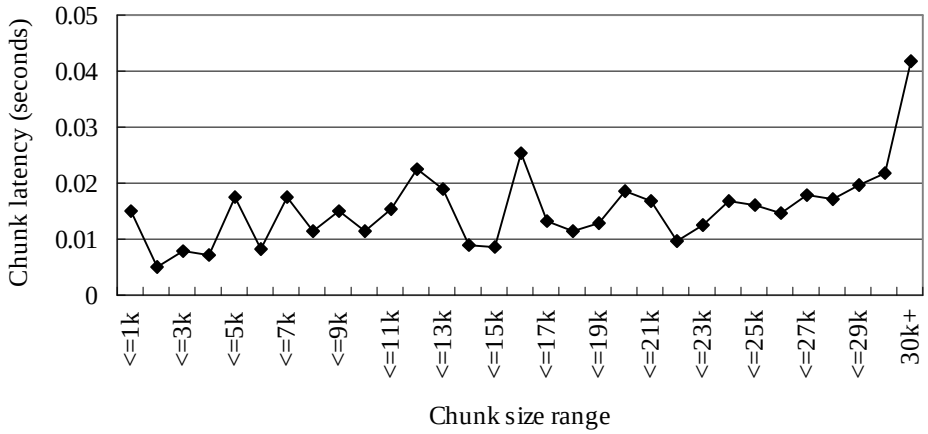


Figure 6.8 Average latencies for delivering chunks with different sizes

Since chunk transfer time contributes towards object fetch latency, we would like to investigate the transfer time for chunks with different sizes. Figure 6.8 plots the transfer time of chunks with respect to their sizes. In our experiments, all chunks were sent out from the same server and delivered over the same distance to the same client, so the latencies of different chunks are comparable.

From Figure 6.8, we see that the distribution of the latencies for chunks with different sizes is quite random. Chunk size does not seem to have much influence on chunk latency. The latency for smaller chunks is often comparable to that of much bigger chunks. This could be due to the random nature of network and server workload. This observation is important because it indicates that mechanisms which reduce chunk size may not help much in reducing object fetch latency.

There are two extreme phenomena in Figure 6.8 worth of mentioning. One is that the latency for very big chunks (those with size greater than 30k) is indeed much bigger than that of smaller chunks. The other phenomenon is that, the latency for the "<=1k" group is even bigger than that of "<=2k" to "<=4k" groups. Further study reveals that this could be due to the TCP slow-start effect as "<=1k" chunks tend to be the first a few chunks in the chunk transfer sequence.

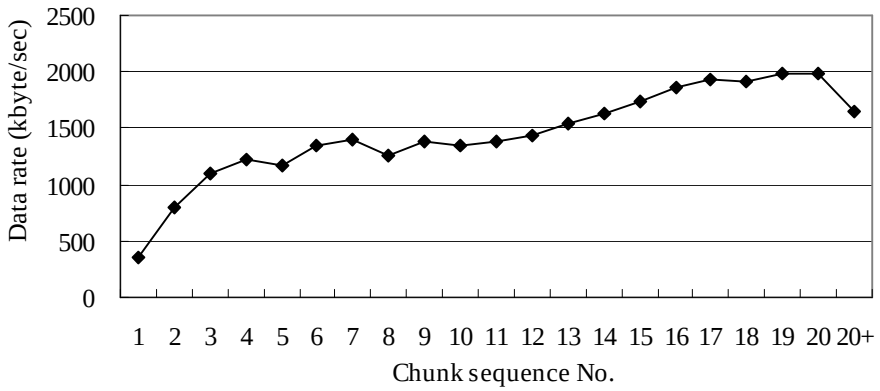


Figure 6.9 Distribution of data rate w.r.t. chunk sequence number

Figure 6.9 tries to explain the TCP slow-start effect by plotting chunk data rate with respect to the chunk sequence number. From this graph, we see that the data rate for the first chunk is significantly much lower than that of later chunks. This indicates that the first a few chunks on a TCP connection are relatively more expensive than the rest, which reflects the TCP slow-start effect in delivering web objects. This characteristic gives further explanation on why the latency for smaller objects is often comparable to larger objects (see Figure 6.4).

Although chunk data rate is generally on the rising for later chunks, it becomes relatively stable from the 4th chunk onwards. So, when the number of chunks in an object transfer is big, TCP slow-start effect would become less significant as it would be amortized with the transfer of large number of chunks.

From the above analysis, we see that the number of chunks is more important in determining object latency than the size of chunks.

6.3 Page Retrieval Latency

Generally, the unit of web browsing is page. So, the page retrieval latency is most meaningful to web users. When we work on web acceleration mechanisms, we should always study their effects on page latency rather than object latency only. This would give us more meaningful results. In this section, we would like to investigate the relationship between object latency and page latency, and the factors that affect page latency.

6.3.1 From Object Latency to Page Latency

A web page usually consists of a container object (CO) and a number of embedded objects (EO). The page retrieval latency is determined by the interaction of the retrieval processes of both CO and EOs. Before we proceed to the detailed analysis of page retrieval latency using WRDM, we first give precise definition of the retrieval latency of web pages to clarify possible ambiguity.

Definition 6.7: Page Retrieval Latency

The retrieval latency of a web page is defined as the time from the initiation of the request for the container object of the page, interpreting the returned data and triggering derived requests for all the embedded objects of the page, till all object data have returned from server and all resources occupied by the requests for both the container object and the embedded objects have been released.

Just like the argument given previously for object fetch latency, we will consider page retrieval latency as the time for successfully retrieved web pages, i.e. all the objects included in a web page must be retrieved successfully. Premature interruption of any objects in a page is not considered in our study (see previous section for reasons).

Mapping the above definition to an object-level WRDG graph, the page retrieval latency for a page ***Page(k)*** is represented by the "longest distance" path in the graph, where the starting point of the path is the request initiation vertex $\upsilon_{r(k,0)}$ of the container object ***Obj(k,0)*** of the page and the ending point of the path is the latest ending vertex $\upsilon_{e(k,i)}$ of an object ***Obj(k,i)*** in the page where $0 \leqslant i \leqslant o$. In the rest parts of this thesis, we may refer to this path as Page Retrieval Cost Path.

Figure 6.10 gives an example object-level WRDG graph showing the longest distance path for the page retrieval latency for a web page with three EOs. Note that the distance of a path is the total weight of the path, not the number of arcs in the path. The longest distance path may not be the path which has the largest number of arcs.

A web page usually consists of a container object (CO) and a number of embedded objects (EO). From Figure 6.10, we see that page retrieval latency is actually derived from object fetch latency of CO and EOs. However, the mapping from object fetch latency to page retrieval latency is not so direct and straightforward. There is complex relationship between object fetch latency and page retrieval latency.

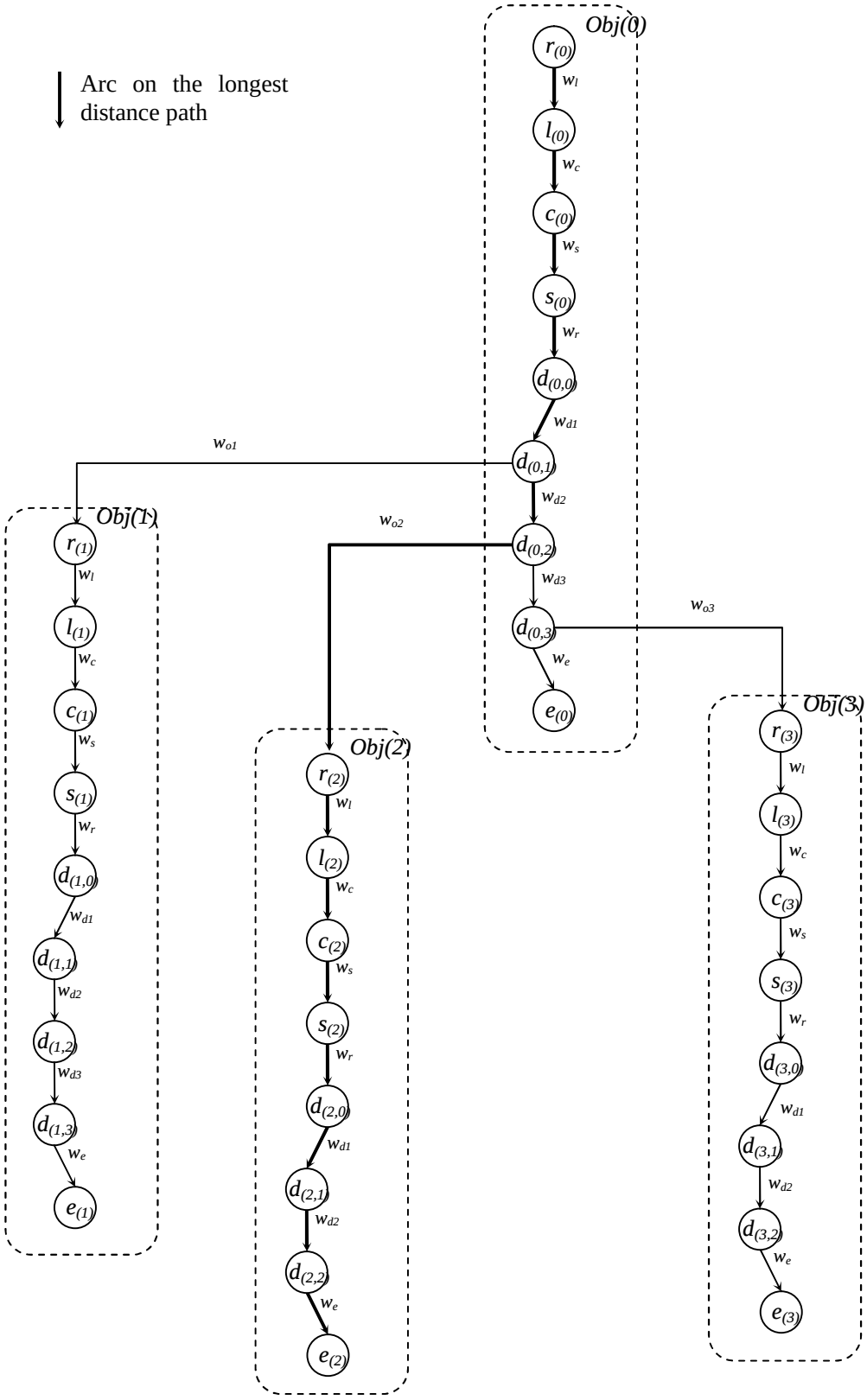


Figure 6.10 Page retrieval latency represented by the longest distance path

First of all, because the URLs of EOs are defined in the CO of the page, so the retrieval processes for EOs highly depends on the retrieval process of CO. The retrieval of an EO can not be started until the CO's data chunk containing the definition of the EO has been transferred from server to the client. This delayed notice of EOs can prolong the retrieval latency for them significantly.

Currently, most common web browsers utilize parallelism for simultaneous fetching of objects in a page. This further complicates the mapping from object fetch latency to page retrieval latency because the overlapping of object latency makes it possible for the fetching of one or more objects to virtually have no effect on the whole page latency. On the other hand, the parallelism width employed in most web browsers is limited, e.g. Microsoft IE and Netscape use a parallelism width of four for the retrieval of objects in a page. With this limited parallelism width, some requests for EOs may be held in waiting state due to the unavailability of parallelism. This waiting time would contribute towards the retrieval latency for the EOs, as well as for the whole page latency.

From the above analysis, we see that object retrieval latency would have at least two more components (in addition to those defined in the previous section) when a group of objects are put together to form a page. The first new latency component is related to the definition of EOs in CO, and we refer to this component as Definition Time of EOs. The second latency component is to reflect the time spent by a request in waiting for available parallelism for retrieval. We call this component Waiting Time of a request. Below we give the definitions of these two new latency components.

Definition 6.8: Definition Time (DT)

Given a page request *Req(Page(k))*, the definition time of an embedded object *Obj(k,i))* in the page is defined as the time from the initiation of the request for the

container object *Req(Obj(k,0))* to the receiving of the data chunk *Chk(k,0,j)* that contains the definition of the embedded object, where $1 \leqslant i \leqslant o$ and $0 \leqslant j \leqslant c$.

In the object-level WRDG graph representing the retrieval process for the requested page *Page(k)*, the definition time of an embedded object *Obj(k,i))* is given by the distance of the path from the request initiation vertex $\upsilon_{r(k,0)}$ of the container object *Obj(k,0)* to a data chunk vertex $\upsilon_{d(k,0,j)}$ of the container object, where the data chunk vertex $\upsilon_{d(k,0,j)}$ has an object deriving arc $a_{o(k,i)}$ connecting to the request initiation vertex $\upsilon_{r(k,i)}$ of the embedded object *Obj(k,i)* .

Note the following two points about the DT times of objects:

First, DT time does not apply to the CO of a page. Or, we can consider the DT time of the CO is always zero.

Second, the measurement of the DT time of an EO starts from the point where the request for the page is initiated. This is because users perceive page retrieval latency from the point when they initiate the request for a page. So the DT time of an EO should be considered as part of the whole EO's latency although the actual retrieval of the EO starts only when the DT time has elapsed.

Definition 6.9: Waiting Time (WT)

Given a page request *Req(Page(k))*, the waiting time of an object *Obj(k,i)* in the page is defined as the time from the time when the existence of *Obj(k,i))* is made known to a client and a request for this object *Req(Obj(k,i))* is initiated, to the time when the request *Req(Obj(k,i))* gets its turn to get processed by the client system, where $1 \leqslant i \leqslant o$.

In the object-level WRDG graph representing the retrieval process for a requested page *Page(k)*, the waiting time for an object *Obj(k,i)* is given by the weight of the object deriving arc $a_{o(k,i)}$ connecting the data chunk vertex $\upsilon_{d(k,0,j)}$ that defines

Obj(k,i) to the request initiation vertex $\upsilon_{r(k,i)}$ for the object *Obj(k,i)* , where $1 \leqslant i \leqslant o$ and $0 \leqslant j \leqslant c$.

For the container object *Obj(k,0)* of a page *Page(k)*, its waiting time is represented by the weight of the page deriving arc $a_{p(k)}$ in a Inter-Page WRDG graph, where the page deriving arc $a_{p(k)}$ connects the ending vertex $\upsilon_{e(k-1,0)}$ of the container object *Obj(k-1,0)* of in the previous page *Page(k-1)* to the request initiation vertex $\upsilon_{r(k,0)}$ of the container object *Obj(k,0)* of the page *Page(k)* .

This WT time exists mainly due to the limited parallelism width for object retrieval. When the number of objects contained in a web page is larger than the parallelism width, the phenomenon of object request being held in waiting would likely occur. Note that the WT time does not apply to the CO of a page either. Or, we can also consider the WT time of the CO is always zero.

The complex relationship between the retrieval processes of objects in a page can be captured by WRDG graphs. Figure 6.11 gives an example object-level WRDG graph showing the retrieval process of a page with five EOs. From the graph, we can clearly see the latency components of the objects and the complex relationship between the objects due to dependency and parallelism. The retrieval processes of the EOs can be started only when their definition is made known to client. Since Obj(5) is defined in the seventh data chunk of Obj(0), its request can not be triggered until that chunk has been returned from server to client. On the other hand, the availability of parallelism also affects the retrieval process of EOs. For example, when the request for Obj(5) is ready for triggering, all the retrieval channels are occupied by other requests. Therefore, the request for Obj(5) has be to be held in waiting state until a retrieval channel become available. This waiting time WT of Obj(5) definitely prolongs its retrieval latency.

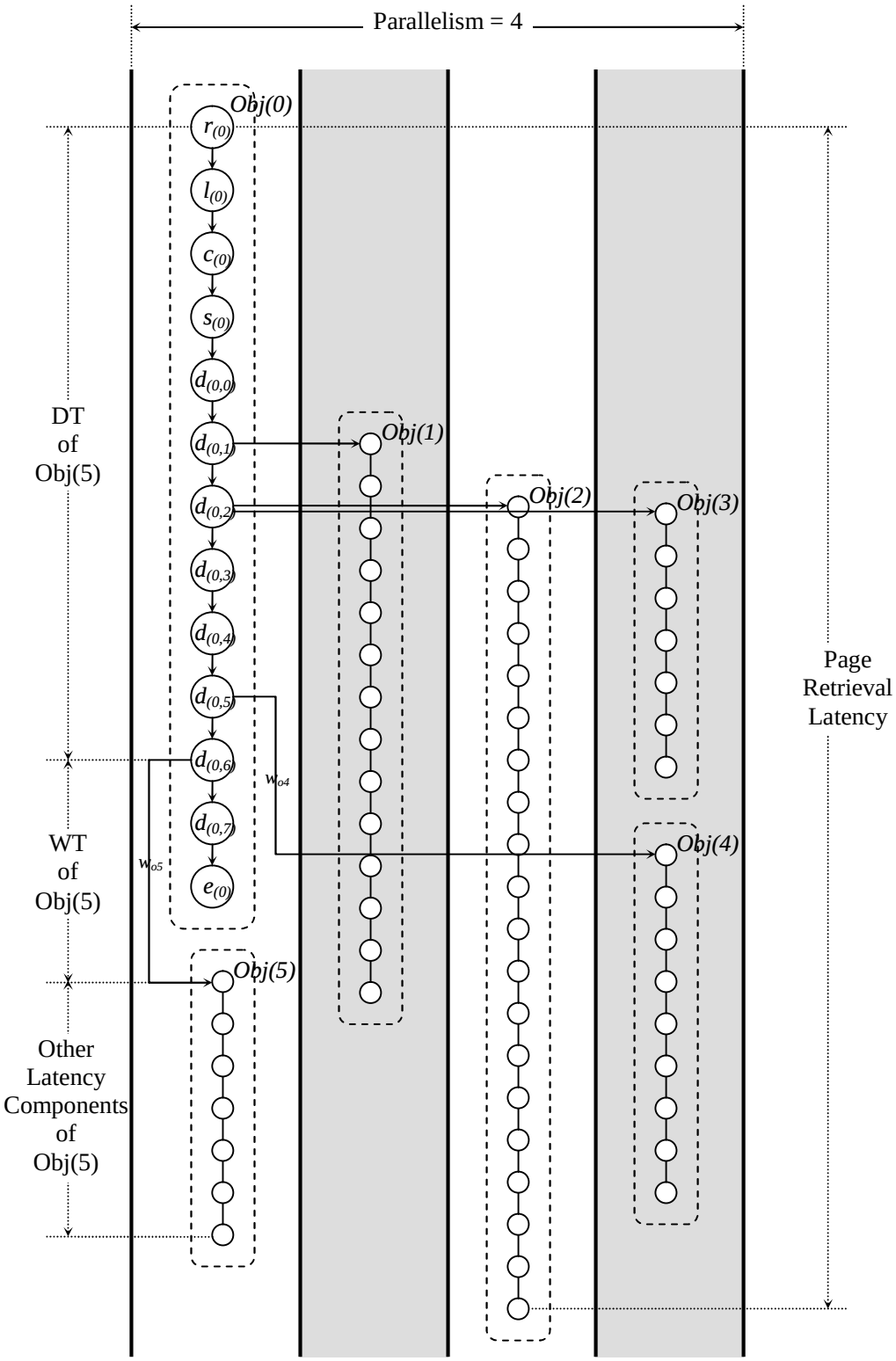


Figure 6.11 Retrieval process for a page with five EOs

Note: Due to space limitation, we simplified the drawings of this graph.

For convenience purpose, here we would like to define two terms to refer to the object latency for individual single objects and for objects in a page. For an individual single object, its latency is made up of the five latency components defined in Section 6.2, i.e. LRT+CT+RST+CST+ET. We will refer to this latency as *Object Fetch Latency (OFL)* from this point onwards in this thesis. In this section, we know that the latency for an object would include two more latency components, namely DT time and WT time, when the object is put into a page. We will refer to the latency which includes DT, WT and OFL of an object as *Object Retrieval Latency (ORL)* in the rest part of this thesis.

In Figure 6.11, the OFL latency of Obj(5) is marked by "Other Latency Components of Obj(5)", and the ORL latency of Obj(5) would include "DT of Obj(5)" and "WT of Obj(5)" on top of OFL latency. Later, when we need to differentiate these two types of latencies of objects, we would call them by different terms, as defined here.

6.3.2 Experimental Study and Analysis

6.3.2.1 General Study

Since web pages are made of objects and page latency is derived from the latency of the objects in the page, we would like to look at the number of objects comprised in pages. Figure 6.12 plots the distribution of pages against the number of EOs per page. We see that while about 9% of pages do not contain any EOs, the majority of pages are made of multiple EOs. A prominent distribution is that nearly 25% web pages contain more than 20 EOs. On average, a page contains about 13.5 EOs.

Next, we would like to look at page latency again page size and compare it with single object latency with similar size. Figure 6.13 plots the distribution of page latency with respect to total page size. By comparing it with the distribution of object latency

shown in Figure 6.4, we have two interesting findings:

1) For sizes below 128 KBytes, page latency is bigger than object latency with the same size. This indicates that there are some factors preventing pages to be retrieved as fast as the objects with the same size. Further study shows that the main reason is the CT time of objects. For pages with sizes smaller than 128 KB, the size of each object in the page should be much smaller than 128 KB (since a page is usually made of multiple objects). According to the results in the previous section, CT time occupies a significant part of object latency when object size is small. So, the overall page latency is prolonged and it becomes even bigger than the latency for objects with sizes below 128 KB.

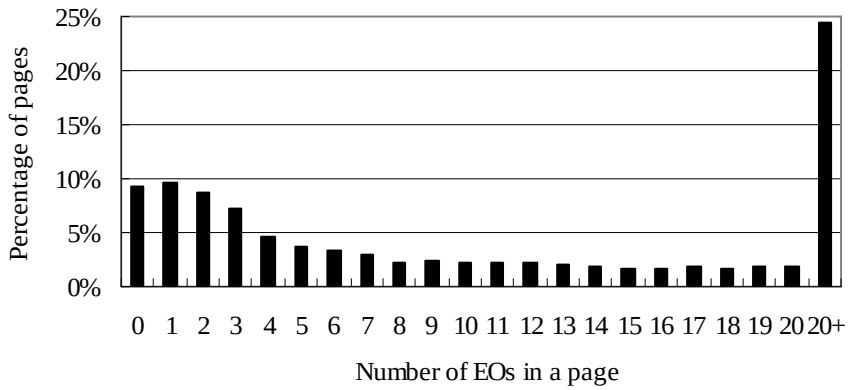


Figure 6.12 Distribution of pages w.r.t. number of EOs per page

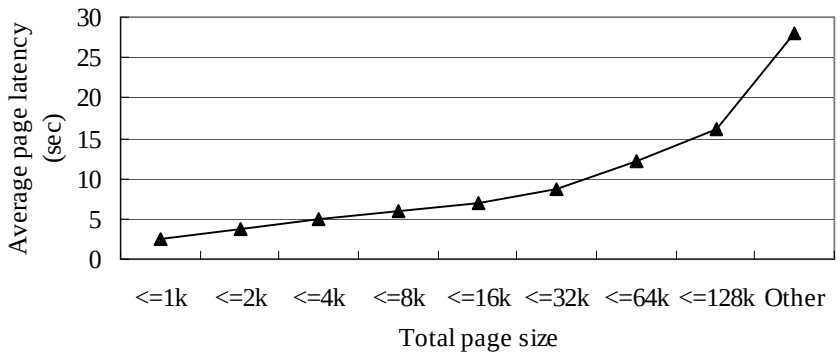


Figure 6.13 Distribution of page latency w.r.t. page size

2) For sizes above 128K, page latency becomes even smaller than object latency with the same size. This shows that the same amount of data can be retrieved faster for pages than objects. Further study indicates that this could be due to the parallel

fetching of objects in a page. When page size is big, it is likely to contain many objects. Current web browsers would fetch multiple objects in parallel. So, the latency of the objects in the page is overlapped, which results in smaller total retrieval latency for page.

The above two findings showed us the complicated relationship between object latency and page latency. Figure 6.14 further confirms this complicated relationship by plotting the distribution of page latency with respect to the number of objects (including CO) in a page. From this graph, we see that page latency is randomly distributed against the number of EOs in a page. The increase in the number of EOs may not result in the increase in total page latency. For example, the page latency for pages with 10 EOs is even smaller than the latency for pages with 7 EOs. This is most likely due to the parallel fetching of objects, which makes the retrieval of some EOs to have no effects on total page latency. On the other hand, page latency is generally on the rise with the increase in the number of EOs in a page. This could be because of the increased DT times of EOs and the shortage of parallelism when number of EOs is big.

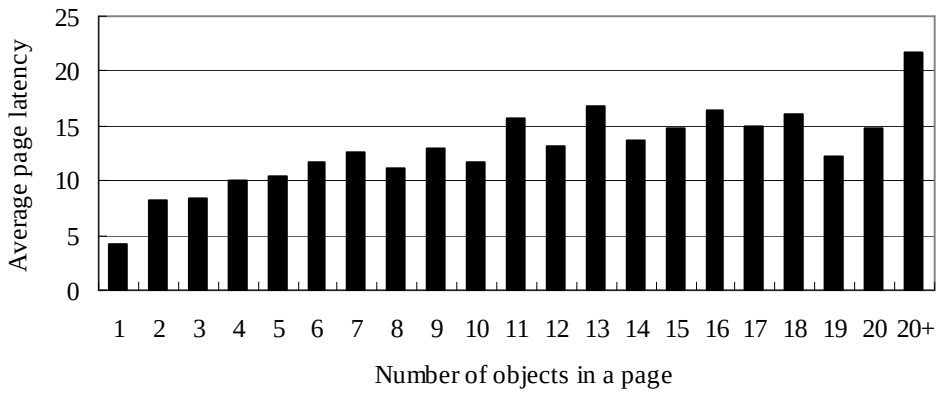


Figure 6.14 Distribution of page latency w.r.t. number of objects in a page

We already see that page latency is determined by complicated factors among objects in the page. According to Figure 6.12, a page contains about 13.5 EOs on average. With so many EOs contained in a page, we may deduce that the impact of the relationship between CO and EOs on whole page latency could be very significant.

Such important factors can not be ignored when studying page latency. In the later part of this section, we will present our study on those special factors particularly found in page retrievals.

As we know from previous section, object retrieval latency would have two new components when objects are put together to form pages i.e. the DT time and WT time of objects in a page. The above discussion indicates there is complex interaction between these two new components and page latency. Now, we would like to investigate the impact of the new components on page latency.

Our experiments recorded detailed information about the latency components of page retrieval. Figure 6.15 plots the relative distribution of the DT time, WT time and the OFL latency of objects against the number of EOs in a page. For a given group of pages (e.g. the pages with 8–11 EOs) in the graph, the relative distribution of these components is calculated using the following formulas:

$$\text{Relative Distribution of DT} = \frac{\sum_{k=1..Num\_of\_pages\_in\_the\_group} \frac{\sum_{i=1..Num\_of\_obj\_in\_Page(k)} \frac{DT\ of\ Obj(i)}{ORL\ of\ Obj(i)}}{Num\_of\_obj\_in\_Page(k)}}{Num\_of\_pages\_in\_the\_group}$$

$$\text{Relative Distribution of WT} = \frac{\sum_{k=1..Num\_of\_pages\_in\_the\_group} \frac{\sum_{i=1..Num\_of\_obj\_in\_Page(k)} \frac{WT\ of\ Obj(i)}{ORL\ of\ Obj(i)}}{Num\_of\_obj\_in\_Page(k)}}{Num\_of\_pages\_in\_the\_group}$$

$$\text{Relative Distribution of OFL} = \frac{\sum_{k=1..Num\_of\_pages\_in\_the\_group} \frac{\sum_{i=1..Num\_of\_obj\_in\_Page(k)} \frac{OFL\ of\ Obj(i)}{ORL\ of\ Obj(i)}}{Num\_of\_obj\_in\_Page(k)}}{Num\_of\_pages\_in\_the\_group}$$

The most surprising finding from the graph might be that a great percentage of retrieval latency of objects in pages comes from DT rather than OFL latency which is often thought of as the dominating factors of page retrieval latency. In all situations, DT time is the largest component among all the latency components. It often takes up

more than 50% of the object retrieval latency. This indicates that DT is a more important latency component than others. This finding gives a hint on a new direction of acceleration of web retrieval latency by reducing the DT times of objects in pages. As we can see from the WRDG graph in Figure 6.11, DT times of EOs exist because the definitions of EOs are found in the data chunks of CO's transfer. The later chunk contains the definitions, the larger the DT times would be. If the definitions of EOs can be made known to client earlier through some special mechanism, the DT time would be reduced, which would in turn effectively reduce the whole page latency. In Chapter 8, we propose a new mechanism to address this issue.

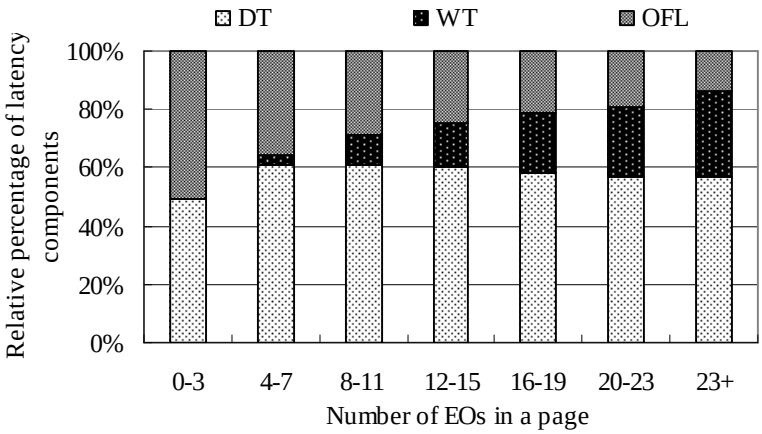


Figure 6.15 Relative distribution of latency components w.r.t. number of EOs per page

Note that in Figure 6.15, although the relative percentage of DT seems dropping with the increase in the number of EOs in a page, its absolute value is actually on the rise. The dropping happens because other components such as WT become bigger, thus it makes DT become relatively smaller. As the number of EOs in a page increases, the absolute DT times generally become bigger. This could be because that more EOs tend to be defined at the middle and bottom part of the CO when there are more EOs in a page.

Another notable finding is that the WT is also a major latency component when the number of EOs in a page is greater than 3, and the relative percentage of WT grows

quickly to as high as 29% of the overall retrieval latency as the number of EOs per page increases. On average, WT time occupies about 15% of the whole retrieval latency of EOs.

The WT time exists because of the limited parallelism width available for object retrieval. In the current web system, most common web client programs such as Microsoft IE and Netscape use a default parallelism width of 4 for parallel fetching. When a page consists of less than 4 EOs, object requests do not need to wait as there is always enough parallelism for every object request upon its triggering. So the WT for such web pages is zero. However, when the number of EOs increases, the WT will increase quickly as the default parallelism width becomes insufficient to handle all the object requests simultaneously. Many object requests would have to wait long for fetching channels to be released by other requests before they can get their turned to be processed. When the number of EOs in a page is greater than 16, the WT time even becomes bigger than the actual object fetch latency OFL. This finding is also very important because it suggests that providing sufficient parallelism width for web retrieval would be an effective way to improve web retrieval latency.

The actual object fetch latency OFL (which consists of LRT, CT, RST, CST, and ET) is often much smaller compared with DT times. As the number of EOs in a page increases, the contribution of OFL to the ORL latency drops dramatically from about 50% to only 14%, even smaller than the WT portion. As a large percentage of web pages contain more than 20 EOs and the average number of EOs in a page is about 13.5 (see Figure 6.12), we expect OFL to be less of a problem for web retrieval than DT and WT for the majority of web pages.

From the above discussion, we are further confirmed about the complicated mapping relationship between object latency and page latency. The object fetch latency

OFL consists of five components and they are the actual latency for fetching an object. However, when the OFL is put into the context of pages, it becomes insignificant. Instead, two other latency components particularly found in pages, i.e. DT and WT, become the dominating factors. In the following subsections, we will study on these factors in detail to further understand their impact on page latency.

6.3.2.2 Studies on DT

Since DT time exists because EOs are defined in CO, we would like to first investigate some distributions of COs.

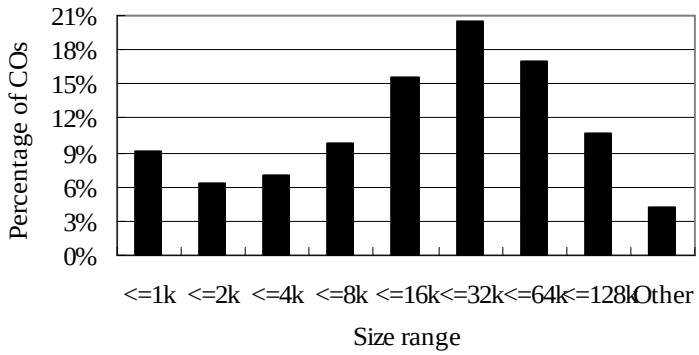


Figure 6.16 Distribution of the size of COs

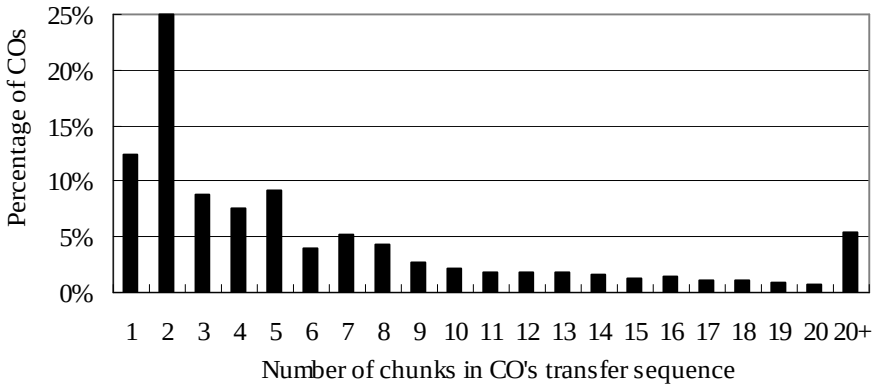


Figure 6.17 Distribution of CO w.r.t. number of chunks

Figure 6.16 plots the distribution of COs with respect to their size. Different from the size distribution of all web objects shown in Figure 6.3 where the majority of objects have sizes less than 8 KBytes, we see that a large percentage of COs have bigger sizes between 8 KBytes to 128 KBytes, with an average size of 35.5 KBytes. For these relatively larger sizes, it may require more data chunks for the transfer. Figure 6.17

shows the distribution of COs against the number of chunks in their transfer. By comparing it with Figure 6.6, we do see that more COs have larger number of chunks than that for all objects shown in Figure 6.6. While about 12% of COs have only one chunk in their transfer, the majority of COs are made up of multiple chunks. On average, a CO has about 6.7 chunks in its transfer. Because COs usually have multiple chunks in their transfer, the chances for EOs to be defined in chunks with large chunk sequence number would be very high, and that would result in large DT times for EOs.

Figure 6.18 through Figure 6.20 plots the distribution of definition points of EOs from three different aspects. Note that the definition points of EOs are actually the starting points of object-deriving arcs in WRDG graphs.

Figure 6.18 shows the average number of EOs defined in each part of CO in terms of the percentage of CO's body size. From this graph, we see that the definitions of EOs are quite evenly distributed throughout CO's body. In other words, every part of CO's body would have EOs defined in it. Thus, some EOs are made known to client quite late until the bottom parts of CO being transferred to client.

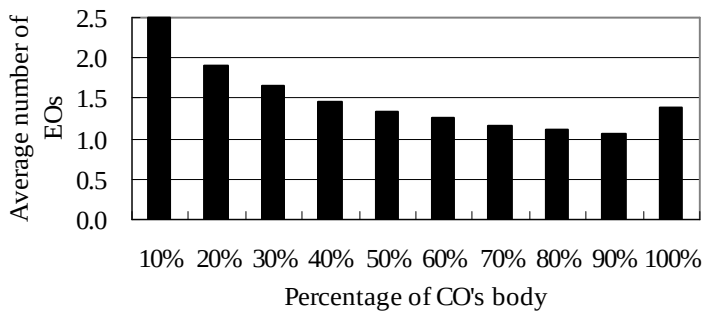


Figure 6.18 Average number of EOs w.r.t. percentage of CO's body retrieved

Figure 6.19 shows the definition points of EOs in terms of chunk sequence number in the CO's transfer sequence. We see that many EOs are defined in the chunks with large sequence number. As the chunks with large sequence number are usually transferred to client at late times, it gives one explanation on why DT times are so

significant in the retrieval latency of EOs.

Figure 6.20 shows the definition points of EOs in terms of percentage of CO's retrieval latency. It is surprising to see that a great many of EOs are defined at the very late parts of CO's retrieval latency. Detailed analysis reveals the following reason: CO's transfer latency is made up of many components (see Chapter 4 and previous sections of this chapter). Most of the latency components such as CT time etc take place before the actual transfer of data starts. So, when the actual transfer of data chunks of CO starts, there is already much time elapsed. That is why we see many EOs are defined at the very late parts of CO's retrieval latency.

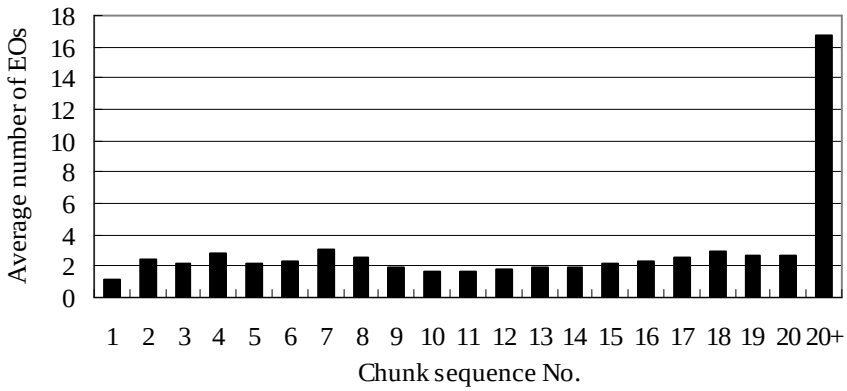


Figure 6.19 Average number of EOs w.r.t. chunk sequence number in CO transfer

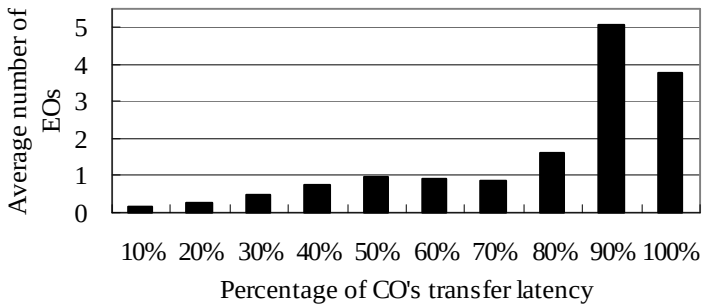


Figure 6.20 Average number of EOs w.r.t. percentage of CO's transfer latency

The above graphs show that a considerable percentage of EOs are defined in the rather later part of COs, which causes large DT times. If the DT times of the EOs are reduced by some mechanism (e.g. shifting the definition of those EOs to earlier locations), will this reduce the effect of DT on whole page latency? To answer this

question, we need to look into whether page latency is determined by EO or CO. Figure 6.21 plots the distribution of EOs according to whether the retrieval processes of them finish before or after CO's retrieval process. According to the graph, we see that most EOs have their retrieval processes finished after CO's retrieval process. On average, about 64% of EOs in a page finish after the CO of the page. In other words, the final retrieval latency of web pages is determined by 64% of the EOs in the page. This indicates that reducing DT times of EOs will have positive impact on whole page latency.

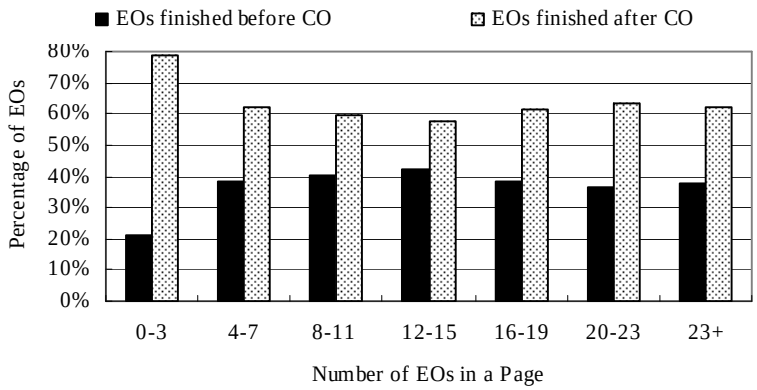


Figure 6.21 Distribution of EOs that finish before and after CO finishes

From the above discussion, we understand why DT time occupies a major portion of the retrieval latency of EOs. Because of this, DT time plays an important role in determining total page latency. Figure 6.22 tries to help us to further understand the impact of DT time on total page latency by showing the effect of four different DT times on total page latency. The four different DT times are defined as follows:

1) DT time = The time of the last chunk of CO

   In this situation, we assume that all the EOs of a page are defined in the last data chunk of the CO. While this situation is less likely to exactly happen in reality, the effect of this situation can be seen in some active web systems. For example, some content transformation systems may require objects to be buffered in the middle of the network for transformation. In the case where the buffered object is the CO of a

page, the definition points for all EOs in that page will be delayed to as late as the time of the last data chunk of the CO in actual fact.

2) DT time = Normal

This situation is the normal situation where the definitions of EOs are distributed across the CO's body as they actually are.

3) DT time = The time of the fist chunk of CO

In this situation, we assume that all the EOs of a page are defined in the first data chunk of the CO. This situation may be rarely seen in current real web systems. However, it is achievable through certain special mechanisms. So, we include it here as a reference.

4) DT time = 0

In this situation, we assume that the definitions of all the EOs of a page are already known to client when the client triggers the request for that page. Again, this is not real in current web systems, but we use it as a reference here.

Figure 6.22 shows the relative page latency under different DT with respect to the number of EOs in a page. From the graph, we see that different DT times do have significant impact on total page latency. In general, pushing the definition points of EOs to the last chunk of CO would cause the page latency to increase about 10.7%, while promoting definition points of EOs to earlier location would result in reduction in page latency for 3.5–10.6%, as compared against the normal situation.

At first, we speculate that reducing DT time may bring in much greater improvement for pages with more EOs since such pages have more EOs to enjoy the reduced DT time. However, we note from Figure 6.22 that the improvement does not seem to grow constantly as the number of EOs in a page increases. Further study reveals that the limited parallelism for parallel fetching of objects could be the reason.

After DT time of EOs has been reduced, parallelism width will become a performance bottleneck. This is because that smaller DT time puts a higher demand on parallelism since EOs are made known for retrieval earlier. Therefore, parallelism becomes insufficient even for pages with smaller number of EOs.

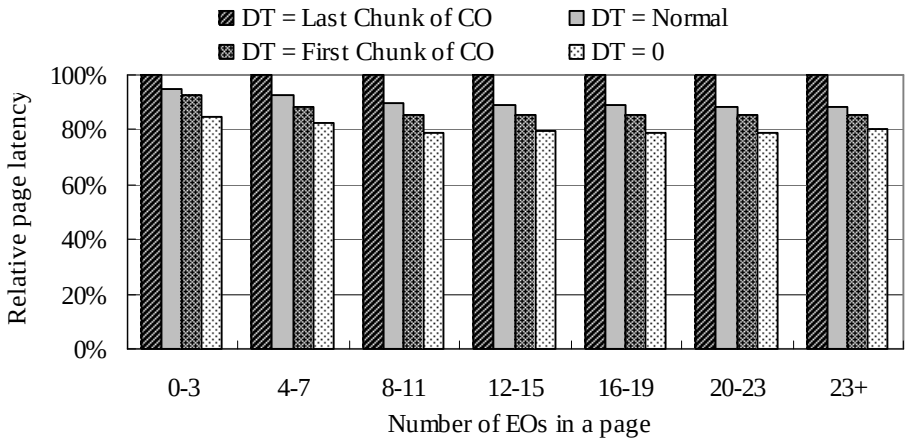


Figure 6.22 Relative page latency under different DT w.r.t. number of EOs in a page

6.3.2.3 Studies on Parallelism and WT

In this subsection, we would like to study the effect of parallelism on total page latency.

In our WRDM model, the WT time of EOs reflects the effect of parallelism on page latency. When the default parallelism width is insufficient for object requests, some requests will be held in waiting state. The time spent by a request in waiting state is captured by the latency component WT time in our model.

According to Figure 6.12, web pages contain about 13.5 EOs on average. Some 25% web pages even contain more than 20 EOs. However, most current web browsers like Microsoft IE and Netscape employ a default parallelism width of four only. As a result, the WT time of EOs is often seen in current web retrieval.

Now, let us first investigate the distribution of EOs being held in waiting state under the most common parallelism width of four. From the aspect of waiting state, EOs can be classified into the following three classes:

Class 1) EOs that are not held in waiting state

For EOs in this group, the contribution of their retrieval latency to page latency will not be affect by an increase in parallelism width. However, a decrease in parallelism width may affect their contribution because decreased parallelism may turn them into waiting EOs.

Class 2) EOs that are held in waiting state, but their retrieval processes finish before the retrieval process for CO finishes

For EOs in this group, their retrieval latency will not contribute to page latency since their retrieval finishes before CO's retrieval process.

Therefore, an increase in parallelism width would not affect their contribution to page latency. But a decrease in parallelism width may change this situation.

Class 3) EOs that are held in waiting state, and their retrieval processes finish after the retrieval processes for CO finishes

For EOs in this group, an increase or a decrease in parallelism width would all affect their contribution to page latency.

Figure 6.23 plots the distribution of EOs belonging to the three classes under the parallelism of four. From the graph, we see that a considerable percentage of EOs are held in waiting state due to the lack of parallelism. Especially, the percentage of EOs belonging to class 3 grows dramatically to over 60% as the number of EOs in a page increases. This indicates that parallelism would have important effect on page latency.

The increase in parallelism width could bring down WT times of EOs, and this reduction in WT times for EOs belonging to class 3 would have positive effect on whole page latency.

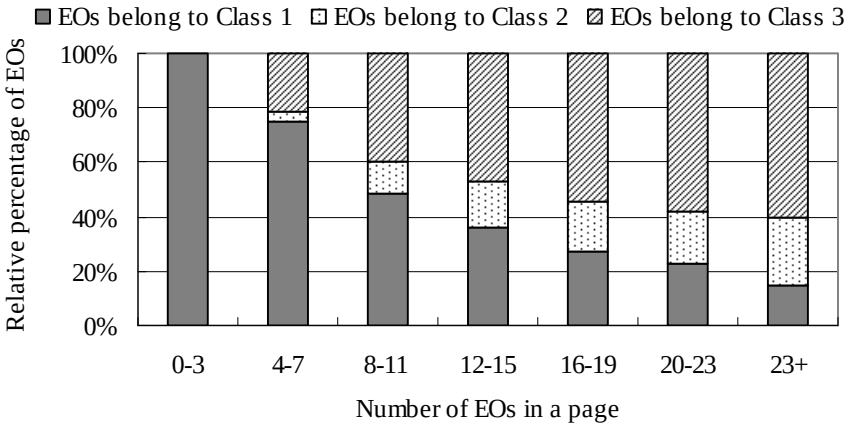


Figure 6.23 Distribution of EOs in waiting state (parallelism = 4)

Figure 6.24 shows the effect of different parallelism width on the distribution of EOs belonging to class 3. As we can see, the percentage of such EOs drops rapidly as the parallelism width increases. When parallelism width grows to 32, the percentage of EOs belonging to class 3 drops to nearly zero percent. Considering that web pages contain about 13.5 EOs on average (see Figure 6.12), it is understandable why almost no EOs are held in waiting state when parallelism width is 32, which is bigger than the number of EOs in most pages.

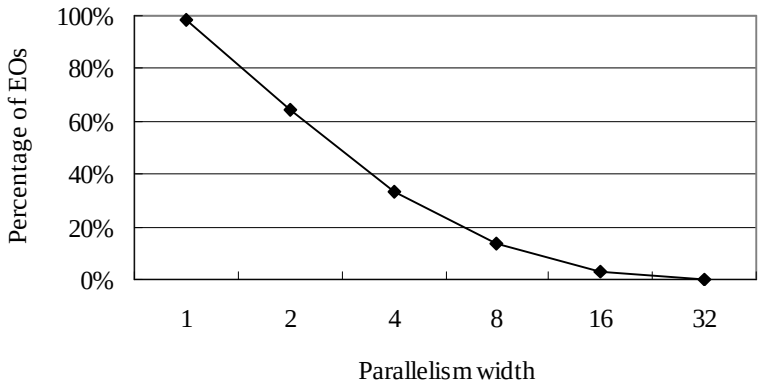


Figure 6.24 Effect of different parallelism width on the distribution of EOs belonging to class 3

Recall in Figure 6.21, the final page retrieval latency is largely determined by the EOs in the page. So we can deduce that improving parallelism width would also have positive impact on whole page latency since wider parallelism can effectively reduce the WT times of EOs. Figure 6.25 shows the effect of different parallelism width on

whole page latency. In the graph, we also include a situation of "parallelism=infinite". When parallelism is infinite, the WT time of objects will be zero. This is the upper bound of the effect that parallelism can bring in on page latency. We use this situation as a reference to compare others against with.

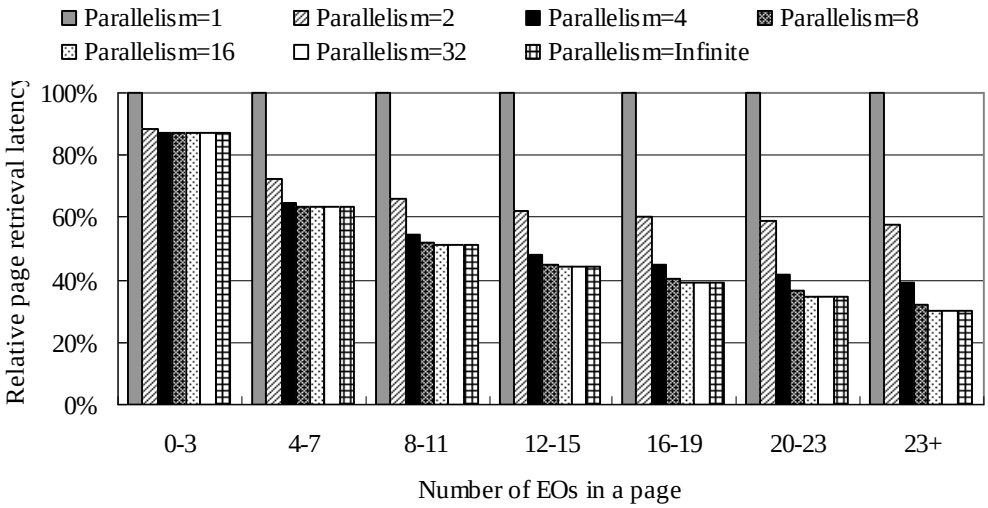


Figure 6.25 Relative page latency under different parallelism w.r.t. number of EOs in a page

This graph confirms the importance of parallelism's effect on page latency. From it, we see that the increased parallelism width would reduce page latency considerably, and this effect becomes stronger when the number of EOs in a page increases. This is understandable because most web pages contain quite many objects (see Figure 6.12). However, when parallelism width grows bigger than 16, the improvement becomes insignificant. This is because the web pages in our test set contain about 13.5 EOs on average (see Figure 6.12). When parallelism width grows bigger than that number, there will be fewer web pages which can take the advantage of the wider parallelism. So the improvement becomes small.

6.3.3 Discussion on the Relationship among DT, WT and Parallelism

From the above analysis, we see there is complex relationship among DT, WT and parallelism, which greatly complicates the mapping of object latency to page latency.

The DT time of EOs exists because the retrieval processes of EOs depend on the retrieval process of CO. Because the definitions of EOs exist in CO, so the triggering times of requests for EOs are highly dependent on when the data chunks of the CO containing the definitions have reached client. Due to the fact that most web pages contain multiple objects and most COs consists of multiple data chunks in their transfer, the DT time of EOs are often considerably large, which contributes significantly towards whole page latency.

The WT time of EOs is caused by insufficient parallelism width for parallel fetching of objects. The parallelism width used in current web client programs is limited, e.g. four for Microsoft IE and Netscape. When the number of objects known for retrieval exceeds the parallelism width, some objects would have to wait until there is free parallel channel for use.

In general, reducing DT time of EOs will put higher demand on parallelism width. When the DT times of EOs are made smaller, more objects will be made known for retrieval at a faster speed. This will require a wider parallelism width. Otherwise, we would only see that more EOs are held in waiting state when their DT times are reduced.

On the other hand, wider parallelism width can only be well utilized when the DT times of objects are small. When parallelism width is increased, it will need more concurrent requests to use up the parallelism. More concurrent requests require more objects to be made known for retrieval earlier, which means smaller DT time of objects. If parallelism width is increased but there are not enough concurrent object requests to take the advantage of it, many parallel channels will just stay in idle, and the resource is wasted.

Furthermore, both DT time and parallelism have something to do with the

number of EOs per page.

Generally, when parallelism width is increased, not only it will require smaller DT of EOs, but also it will require more EOs to be defined in the page so that there may be more concurrent requests to make use of it. But if there is too large number of EOs included in the page, many of the objects may suffer from long WT times due to the relative insufficiency of parallelism. For example, the studies in previous sections show that the default parallelism of four in current web system seems insufficient since current web pages consist of 13.5 EOs on average. Given the trend that web pages tend to have more objects, a wider parallelism should be considered.

Conversely, while larger number of EOs in a page is good to wider parallelism, it may mean bigger DT times for EOs because the CO of the page also tends to be big in this situation. Unless special mechanism is taken to reduce the DT time, otherwise, the increase in the number of EOs per page will prolong page retrieval latency.

In brief, DT or parallelism will become performance bottleneck when the other one is improved, and they have contrary requirement on the number of EOs in a page. Because the interaction among the factors is so complex and all the factors are very critical in determining page retrieval latency, so it requires prudent consideration of all the factors in order to achieve optimal web retrieval performance. Simply adjust any one of them will not bring in the best improvement because other factors will soon become performance bottleneck if only one is improved.

6.4 Impact of Real-time Content Transformation on Web Retrieval Latency

6.4.1 Real-time Transformation of Web Content

With the exponential growth of web usage, web has become the most important and popular communication media in the world. Everyday, millions of people access the web from every corner of the world using different types of devices such as PCs,

cellular phones and PDAs etc. Due to different preference of users and different environment such as network bandwidth and capability of users' devices, different group of users may have different expectation on the presentation of the content they surf. For example, some web users may expect the content to be in their native language, while some others may expect to download the key content to their mobile devices fast and ignore some trivial content like unimportant images. To cater for the different needs, there emerges a technology called content transformation. This technology tries to transform web content to best satisfy the different expectation of users, since this is very important to both web users and web content providers.

Web content transformation is often done on web intermediary servers like proxies. This is because such solution has many advantages. First, this solution is cost-effective as it uses dedicated hardware design for content transformation and delivery and it has the one-to-many nature. Second, the management is centralized in such solution so it is easy to manage the system. Third, this solution is easy to deploy because there is no need for collaboration from web servers and clients. Finally, there are some types of content transformations which should be done more appropriately in the intermediary servers. Examples of such transformation include advertisement localization and content personalization etc.

Content transformation on web intermediary servers is often carried out in real-time because the intermediary servers usually do not have all the content to apply offline transformation on it. When doing real-time content transformation, web intermediary servers basically have the following three ways to perform the task:

1) Chunk-streaming approach

   As we know, web content is transferred in a sequence of data chunks from server to client. In this approach, web intermediary server will apply content transformation

on each data chunk it receives and then forward it to client. This way, transformation is done on the fly, without delaying the transfer of each chunk.

2) Partial-object buffering approach

In this approach, web intermediary server will buffer certain number of data chunks before it apply content transformation on them. This is important for some transformations which require some previous or/and future data to perform the transformation. After transformation, some data chunks are forwarded to clients while some may be kept for the transformation on the following data.

3) Full-object buffering approach

This approach buffers the whole object at web intermediary server and then perform the necessary transformation on the whole object. After that, the object will be forward to clients, still in a chunk-by-chunk way.

While many studies focus on the real-time feature and the restrictions on the kind of transformation that can take place etc. for the above three approaches of content transformation, there is little study on the possible impacts of these different content transformation approaches on web retrieval latency. In this section, we would like to use our WRDM model to analyze the performance impacts of content transformation, with special emphasis on page retrieval latency.

6.4.2 Impact of Content Transformation on Web Retrieval Latency

In order to study the performance impacts of content transformation using our WRDM model, we first need to extend our WRDG graphs to capture web retrieval process when intermediary servers are in presence. Figure 6.26 gives a WRDG graph for the retrieval process of a page with 2 EOs. In the intermediary server, every data chunk would have two associated operations: receiving from web server and forwarding to client. So we use two vertices to represent one data chunk for the retrieval process in

intermediary server: the vertex υ_{ir} represents the receiving of the data chunk *Chk(i)* from server, and the vertex υ_{if} represents the forwarding of the data chunk *Chk(i)* to client. For simplicity reason, we do not show the retrieval process of EOs in the graph. Instead, we just show the definition points of the EOs.

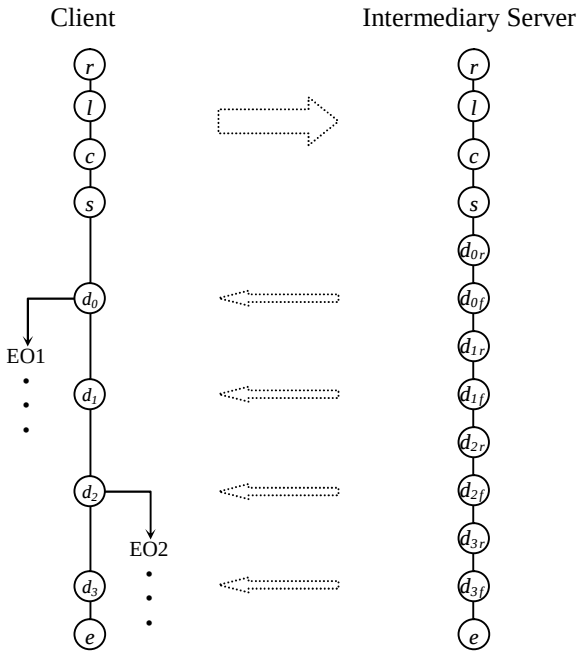


Figure 6.26 WRDG graph for retrieval process in the presence of intermediary server

When chunk-streaming approach of content transformation is carried out on intermediary server, there will be some transformation processing time between the receiving and forwarding of every data chunk. We use a vertex υ_t to represent this transformation operation. Figure 6.27 demonstrates the retrieval process for a page with two EOs when intermediary server applies chunk-streaming transformation on the content. Because intermediary servers often have special dedicated hardware design for content transformation, so the latency incurred by the transformation for a data chunk is often negligible to client. So, we can generally assume that the user perceived latency in Figure 6.27 is the same as the one in Figure 6.26.

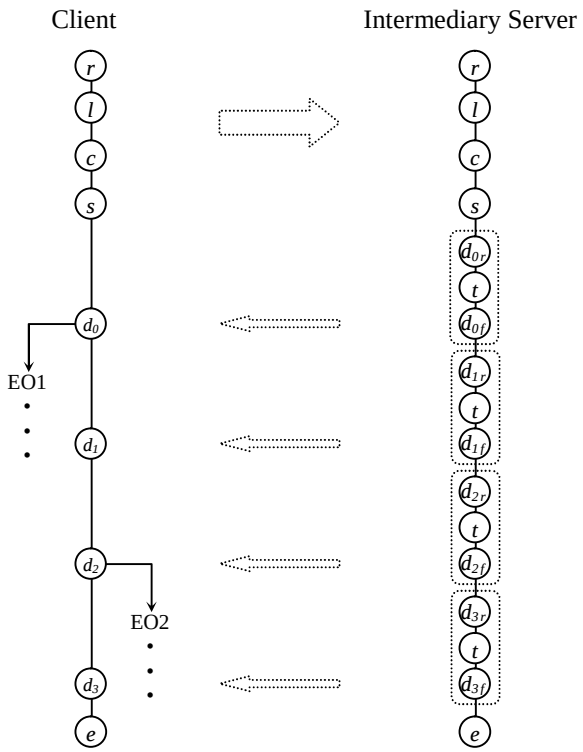


Figure 6.27 Retrieval process for chunk-streaming transformation

When the intermediary server employs the partial-object buffering approach to do content transformation, it may affect the definition time (DT) of EOs. Figure 6.28 gives an illustration of the retrieval process for a page when intermediary server uses partial-object buffering transformation approach. Here we assume the intermediary server always buffers two chunks for content transformation. Comparing this graph with Figure 6.27, we can see that the definition points of both EO1 and EO2 have been postponed to later points. We call this effect as push-back effect of content transformation.

Pushing the definition points of EOs to later points means that the DT times of EOs will become larger. As we learnt in previous sections, DT times of EOs play very important role in whole page latency. Enlarging DT time could result in increase in whole page latency.

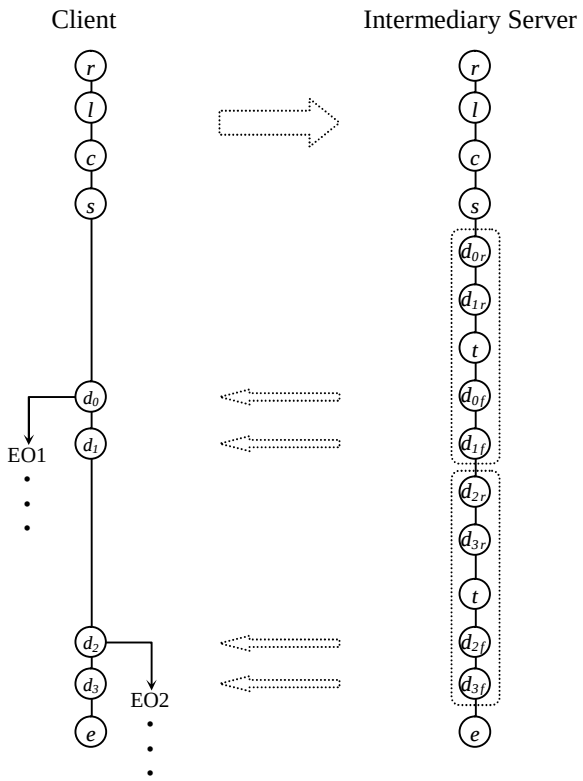


Figure 6.28 Retrieval process for partial-object buffering transformation

Figure 6.29 shows the retrieval process for a page when full-object buffering transformation is used. From this graph, we see that the push-back effect becomes even more serious. Because the intermediary server buffers the whole object before it applies content transformation, so the forwarding of every data chunk has been postponed to the bottom severely. This could dramatically increase the DT times of EOs, which would in turn increase whole page latency.

6.4.3 Experimental Study

We conduct simulation experiments to find out the impacts of real-time content transformation on web retrieval latency. The approaches we examined are the partial-object buffering transformation and the full-object buffering transformation. For the partial-object buffering transformation, we assume the intermediary server always buffers two chunks of data for content transformation. As for the chunk-streaming transformation, we assume its impact is negligible and treat its

performance is the same as the normal situation, i.e. web retrieval through intermediary servers with no content transformation. The normal situation is included in our study to be used as a reference base for comparing the performance of different transformation approaches.

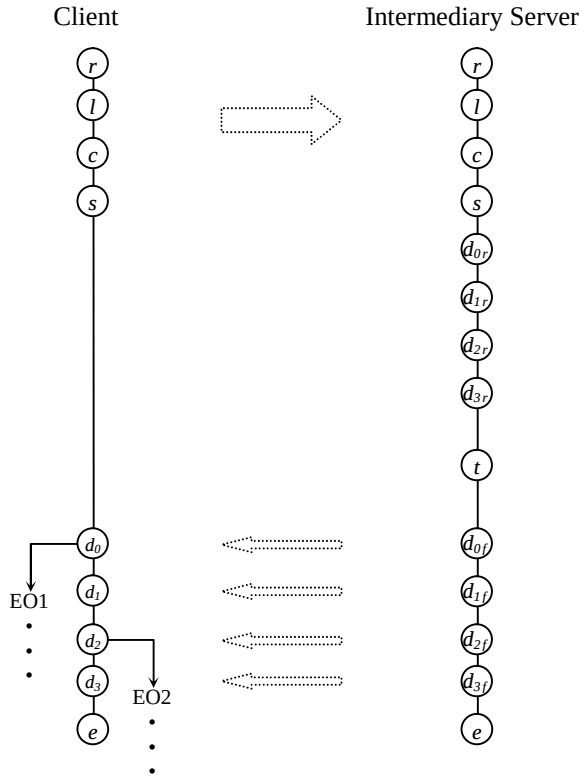


Figure 6.29 Retrieval process for full-object buffering transformation

Figure 6.30 shows the impact of real-time content transformation on DT times of EOs. From it, we can see that content transformation have significant impact on the DT times of EOs. The impact of full-object buffering transformation is much higher than that of partial-object buffering transformation. On average, DT times of EOs have been increased about 63.5% and 18.8% by full-object buffering transformation and partial-object buffering transformation, respectively. With this significant impact, we expect that there would be substantial increase in whole page latency when full-object buffering transformation or partial-object buffering transformation is conducted on intermediary servers.

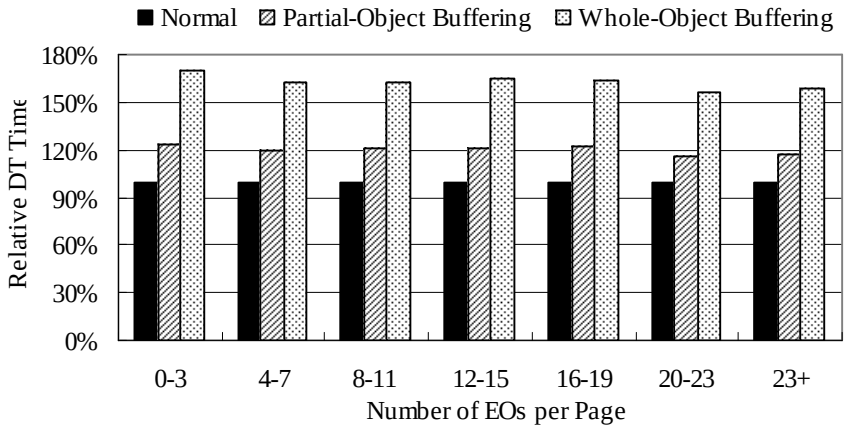


Figure 6.30 Impact of real-time content transformation on DT times of EOs

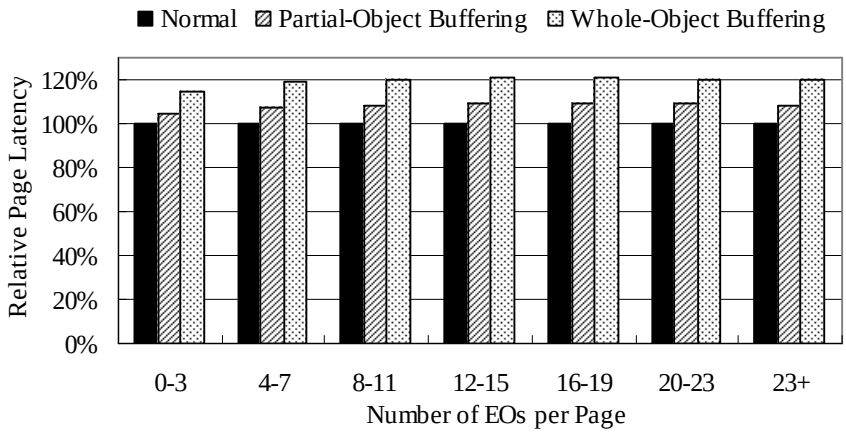


Figure 6.31 Impact of real-time content transformation on page retrieval latency

Figure 6.31 shows the impact of real-time content transformation on page retrieval latency. As expected, we see that they indeed result in substantial increase in whole page latency. When full-object buffering transformation or partial-object buffering transformation is conducted on intermediary servers, page retrieval latency would be increased by 10.6% and 4.9% respectively. We also note that the impact of real-time content transformation on page retrieval latency is not as significant as it does on the DT times of EOs. This could be mainly due to the parallelism in web retrieval, which prevents object latencies from being mapped into page latency directly.

From the above study, we see that full-object buffering real-time content transformation has the most severe impact on web retrieval latency. So we would like to suggest not using it in web systems. The chunk-streaming transformation has the

least impact on web retrieval latency and such impact is often negligible. So it may be considered when implementing real-time content transformation. However, for certain types of transformation which requires seeing more data than just one chunk of data, we would better use the partial-object buffering approach. As we see from the above results, the partial-object buffering approach does have some negative effect on page retrieval latency, but the effect is moderate. So, it should be highly preferred rather than the full-object buffering approach when more data need to be seen for content transformation.

6.5 Upper Bounds of Improvement on Web Retrieval Latency

Many acceleration mechanisms have been proposed to tackle the problem of long web retrieval latency. The reuse-based mechanisms (e.g. web caching and prefetching) are the first category mechanisms being proposed. Many studies have examined the upper bound for the performance of such mechanisms and showed that their improvement is limited [13, 14, 17, 18, 19, 20, 108, 109]. To overcome the limitation, another new direction which aims to accelerate the actual retrieval process of web pages is getting more attention. The new direction is believed to have better potential because they do not suffer from the cacheability related issues such as the low reuse ratio of web objects and the ever-increasing amounts of dynamic web content. Examples of such techniques include compression, content selection and persistent connection, etc. While the new direction of acceleration has shown some promising potential ([22, 23, 24, 25, 26, 27, 28, 37] etc.), it remains to be seen the quantitative upper bound of the techniques in this direction. Below, we would like to investigate the upper bounds for the performance improvements for the acceleration mechanisms in this direction. The results would help us to get some idea about the potentials of this direction of acceleration.

In deriving the quantitative upper bounds, we make some best-cases assumptions about the latency components in web retrieval to base our simulations on. The performance of these best-cases will be the upper bounds for web retrieval under different situations.

6.5.1 Upper Bounds for Location Resolution Related Acceleration

Web retrieval process typically starts with the resolution of the location of a server. In current web system, such location resolution processes are DNS lookups. The latency caused by this process is referred to as LRT in our WRDM model. Many mechanisms have been proposed to reduce the LRT time in web retrieval, such as stored address binding [194], pre-performing DNS lookups [195], proactive caching of DNS records [196] and so on.

We can derive the upper bound for location resolution related mechanisms by assuming the best-case situation for this process. Since this process is typically DNS lookup in current web system, so the best-case situation would be that 100% DNS lookups result in hits in local DNS cache. Figure 6.32 illustrates this best-case situation using a WRDG graph for a page with one EO.

By assuming the smallest LRT time for every request, we can derive the performance upper bound for location resolution related acceleration mechanisms. Figure 6.33 plots our simulation results of the performance of the best-case situation.

From Figure 6.33, we see that the performance of the best-case situation is better than the normal situation, but the difference is not very significant. On average, the best-case situation improves the performance by about 5.07%. This shows that the room for improvement in this direction is rather limited. This is because the location resolution related acceleration mechanisms only reduce the LRT time which occupies just a small portion of the whole object latency. Refer back to Figure 6.5, LRT time

contributes about 1~6% of the whole object latency. This indicates that the performance of mechanisms in this direction will not be significant, although there is still some room for improvement.

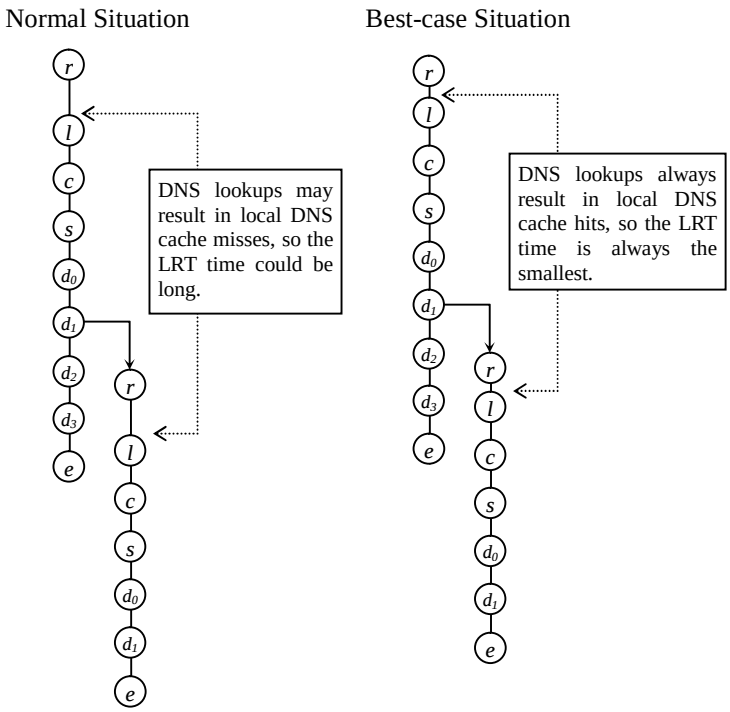


Figure 6.32 Best-case assumptions for location resolution related mechanisms

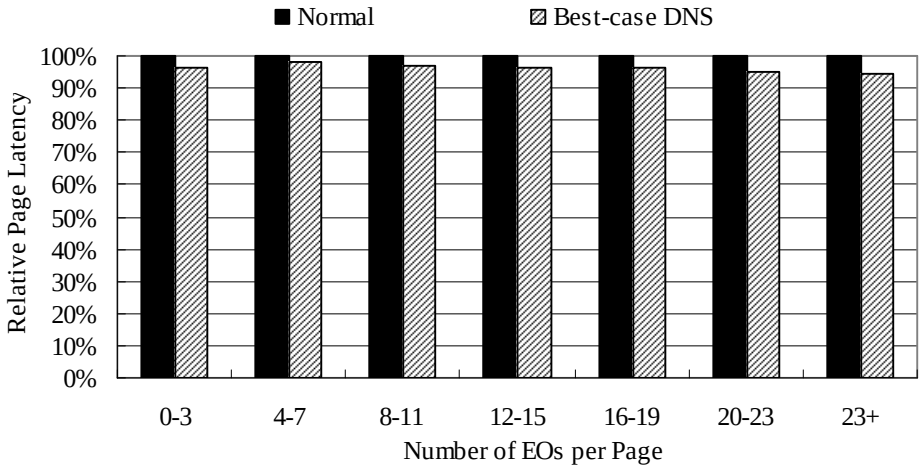


Figure 6.33 Upper bounds for location resolution related mechanisms

6.5.2 Upper Bounds for Connectivity Related Acceleration

The establishment of network connection plays a very important role in web retrieval. The latency incurred by network connection establishment, i.e. the CT time,

is one of the most significant components in object retrieval latency. From Figure 6.5, we already see that CT time can sometimes occupy more than 50% of the whole object latency.

Typical connectivity related acceleration mechanisms include persistent-connection, pre-connection and bundling etc. [21, 22, 24, 192, 195, 197, 198, 199, 200], MGET [22, 23, 25, 201, 202]. All these mechanisms try to accelerate web retrieval by reducing the CT time for every request.

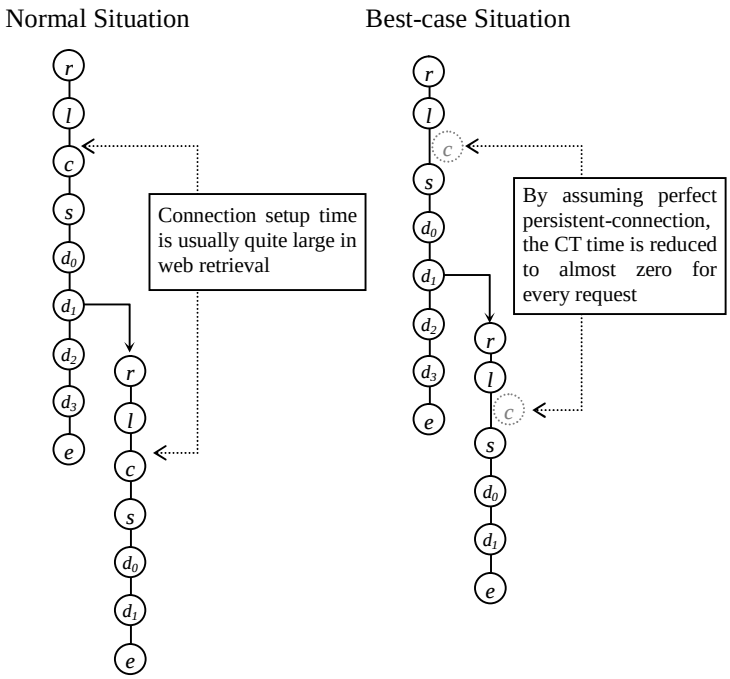


Figure 6.34 Best-case assumptions for connectivity related mechanisms

To derive the upper bound for connectivity related mechanisms, the best-case situation needs to be assumed. In our study, we assume perfect persistent-connection in web retrieval. By perfect persistent-connection, we mean that every request enjoys persistent-connection, so the time spent on network connection establishment will be the minimum. Figure 6.34 demonstrates how this best-case assumption affects page retrieval latency. Using this assumption, we get an upper bound for the mechanisms in this direction, which is shown in Figure 6.35.

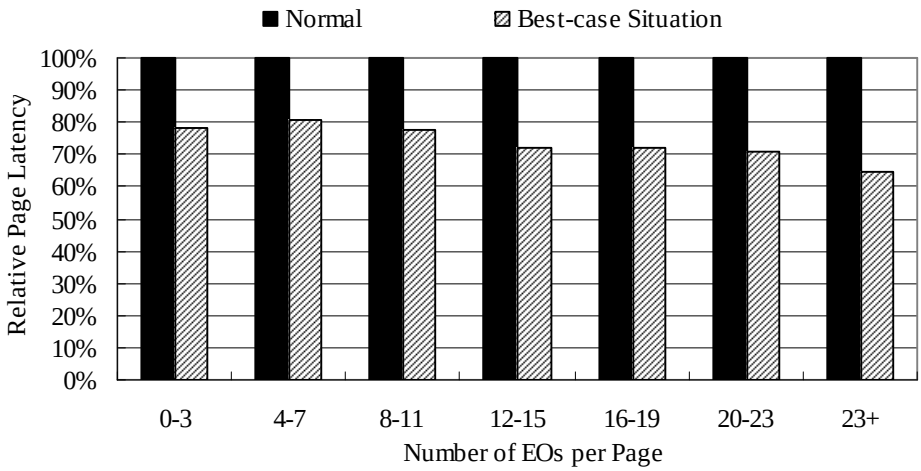


Figure 6.35 Upper bounds for connectivity related mechanisms

From Figure 6.35, we see that the performance gain by the best-case situation is significant, with an average of about 27.64%. This result shows that the room for improvement in this direction is substantial. It also reemphasizes the importance of CT time and the effectiveness of those connectivity related mechanisms.

6.5.3 Upper Bounds for Transfer Related Acceleration

The latency incurred by the actual transfer process is the CST time in web retrieval. As we learnt from Figure 6.5, CST time often makes up more than 50% of the whole object latency. For big objects, it can reach as high as 95% of the object latency.

There are quite a number of mechanisms aiming to reduce this significant latency component. Typical examples include delta encoding, compression, transcoding and content optimization etc. [220, 218, 221, 224, 219, 222, 223, 26, 218, 219, 203, 203, 217, 27, 225, 227, 226, 28, 228, 204]. All these mechanisms try to reduce the amount of data being transferred during the retrieval of an object.

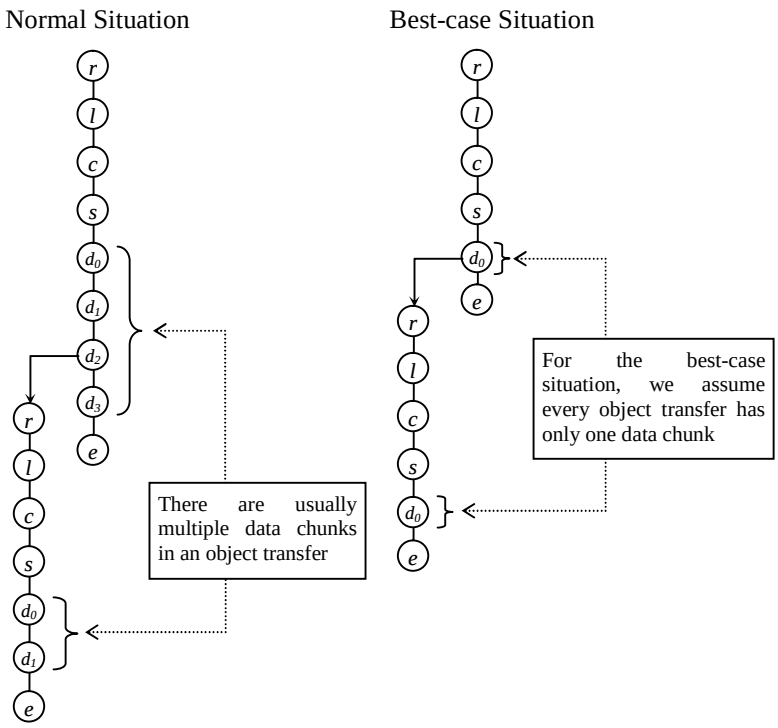


Figure 6.36 Best-case assumptions for transfer related mechanisms

Again, we need to assume the best-case situation in order to derive the upper bound for transfer related acceleration. In our study, we use this situation as the best-case situation: there is only one chunk of data being transferred during the transfer process of any objects. We believe this shall be a reasonable best-case situation as every object request would result in at least one chunk of data being returned from the server. Figure 6.36 gives a simple illustration of this assumption using WRDG graphs. Based on this best-case assumption, we are able to derive the upper bound for the mechanisms in this direction, as shown in Figure 6.37.

From this graph, we see that the performance of the best-case of transfer related acceleration is also significantly better than the normal situation. The average difference between them is about 15.11%. We notice that the difference in performance in Figure 6.37 is generally smaller than that in Figure 6.35. At first, we feel this is somewhat surprising because we know that CST time is equal or even more important to web retrieval latency than CT time. But further study reveals the reason for this

result: For the best-case situation in this study, we assume there is only one chunk of data in the transfer of any objects. With reference to Figure 6.8 and Figure 6.9, we see that the latency for transferring one data chunk is actually quite large. This may explain why the overall improvement is not as significant as the one shown in Figure 6.35.

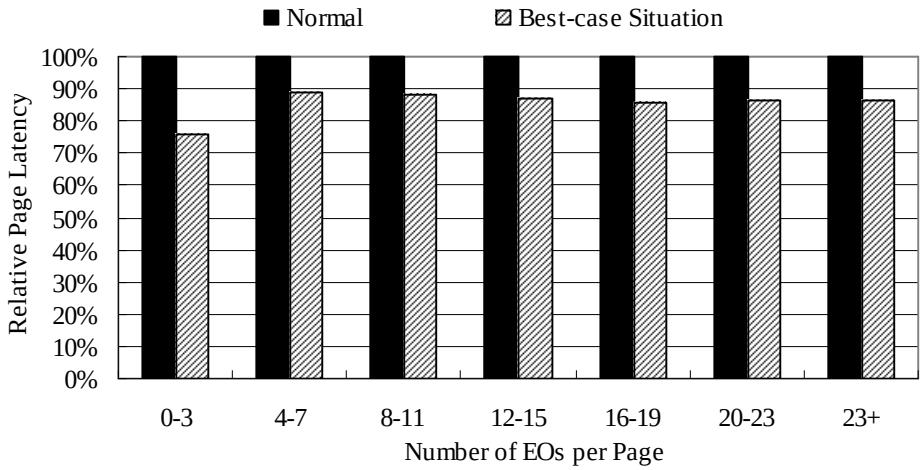


Figure 6.37 Upper bounds for transfer related acceleration

6.5.4 Integrated Upper Bounds for Web Acceleration

Finally, we would like to investigate the integrated upper bounds for web acceleration when the best-case situation is assumed for every possible step in web retrieval. For this study, we include four best cases in our study and derive four upper bounds based on them. The assumptions for these best cases are listed below. Note that the assumptions for the four upper bounds only differ on the parallelism width and the DT times of EOs.

Ÿ Assumption on parallelism width (i.e. WT time of EOs)

We assume two parallelism widths for deriving the upper bounds.

(1) The first parallelism width is four, i.e. the normal situation.

(2) For the other situation, we assume infinite parallelism width is used. Under this assumption, the WT time of all objects will be zero, which is undoubtedly the best case for WT times.

Ÿ Assumption on DT time of EOs

For this parameter, we have two different assumptions:

(1) We assume the DT time of all EOs equal to the time of the first data chunk of the CO of the page. This is equal to say that we assume all the EOs of a page are defined in the first data chunk of the CO. This shall be the best practical case for current web pages;

(2) We assume the DT times of all EOs are equal to zero. This will be theoretically the best case that DT time can be.

Ÿ Assumption on LRT time

We assume the LRT time for all requests is always the smallest. Because the LRT time comes from DNS process in current web system, this assumption means that all DNS processes are treated as local-cache-hit. According to our experimental result, a local-cache-hit DNS lookup typically takes about 0.003 seconds to finish.

Ÿ Assumption on CT time

CT time comes from the establishment of network connection. Here we assume "perfect persistent connection" for our simulation. By "perfect persistent connection", we mean that all requests would enjoy persistent connection, which is even more than the best case that current techniques can deliver. With this assumption, CT time will always be the smallest.

Ÿ Assumption on CST time

CST time is the actual transfer time of data chunks and we assume it is related to the size of objects. For all objects, we try to use the smallest possible sizes for them. In detail, we assume there is only 1 chunk of data transfer for text objects like COs, and for other types of objects like images, video and audio files etc., we assume 1/2 of the original size is used. We believe this shall be reasonably the best sizes that

any encoding, transcoding, or content selection algorithms may give.

There are two other latency components, i.e. RST and ET. For them, no best-case values are assumed and we just use the normal values. This is because they can hardly be reduced in current web system, and their impact on page latency is significantly much smaller compared with other latency components.

The combination of the above assumptions gives four best case situations for deriving the upper bounds. The assumptions for each best case are briefly listed in Table 6.1 and illustrated by the WRDG graphs in Figure 6.38 and Figure 6.39.

Table 6.1 Assumptions for the best cases

	Best Case 1	**Best Case 2**	**Best Case 3**	**Best Case 4**
Assumption on parallelism	Parallelism = 4		Parallelism = Infinite (i.e. WT = 0)	
Assumption on DT	DT = 1st chunk time	DT = 0	DT = 1st chunk time	DT = 0
Assumption on LRT	LRT = Local-cache-hit DNS lookup time (about 0.003 seconds)			
Assumption on CT	Perfect persistent connection (0.031749 seconds)			
Assumption on CST	Text objects: Assume its transfer contains only 1 chunk of data Other types of objects: 1/2 of the original size is assumed			

We used the traces and tools described in Chapter 5 to conduct simulations in order to derive the quantitative upper bounds. The results we got are plotted in Figure 6.40. The normal situation, i.e. normal DT time and parallelism equal to four, is used as a reference against the upper bounds.

From Figure 6.40, we see that all best-cases give much better performance than the normal situation. This indicates that the room for improvement is significant. On average, the best-case situations can improve the performance from about 70% to about 75%. These high percentage improvements of the upper bounds suggest that the acceleration which aims to accelerate the actual retrieval process of web objects and pages is a very promising research direction.

Besides the great potential given by the upper bounds, we also have an

interesting observation from Figure 6.40. We see that the performance of "Parallelism=Infinite" (i.e. Best Case 3 and 4) is better than the performance of "Parallelism=4" (i.e. Best Case 1 and 2), and the difference between them becomes larger when the number of EOs in a page increases.

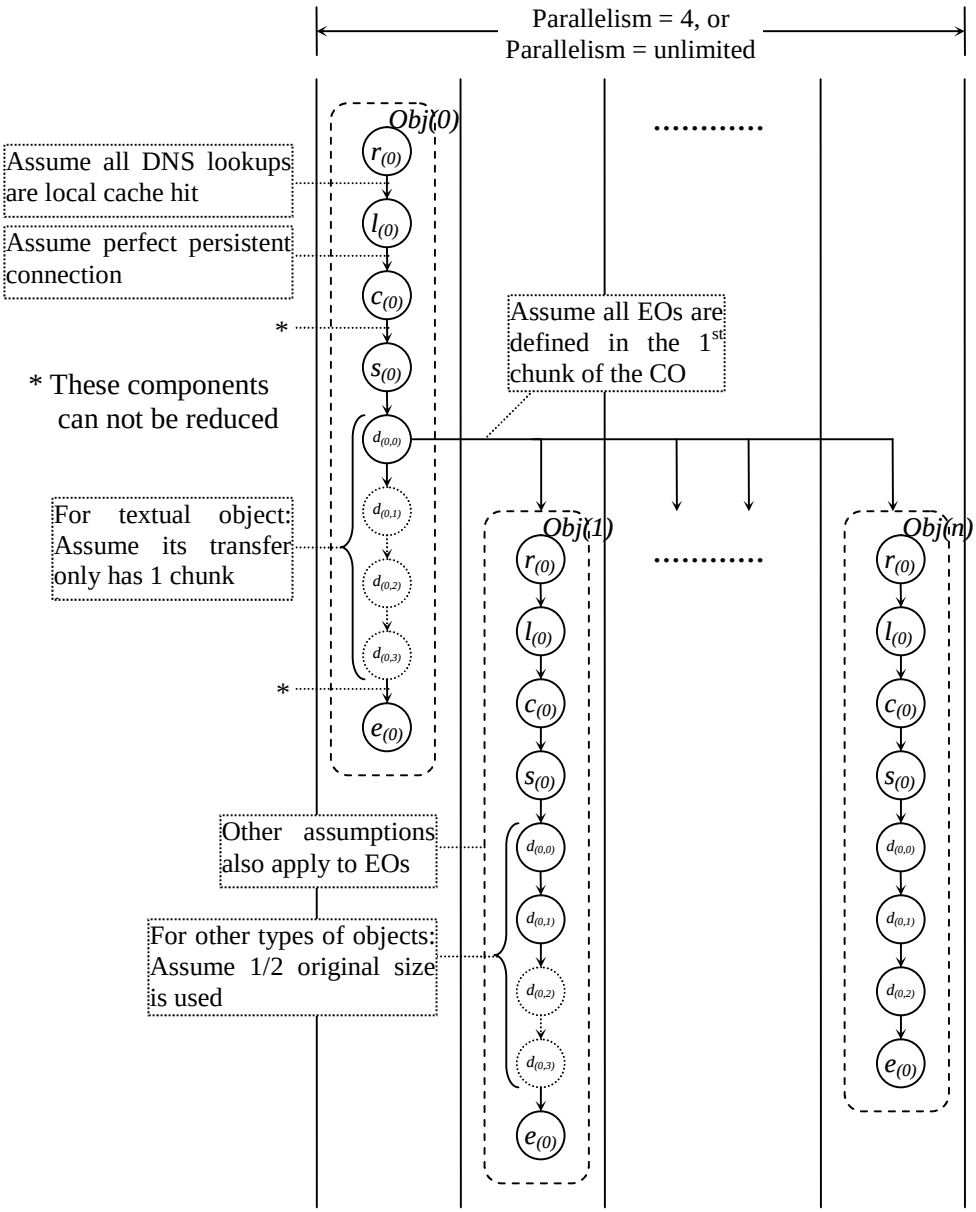
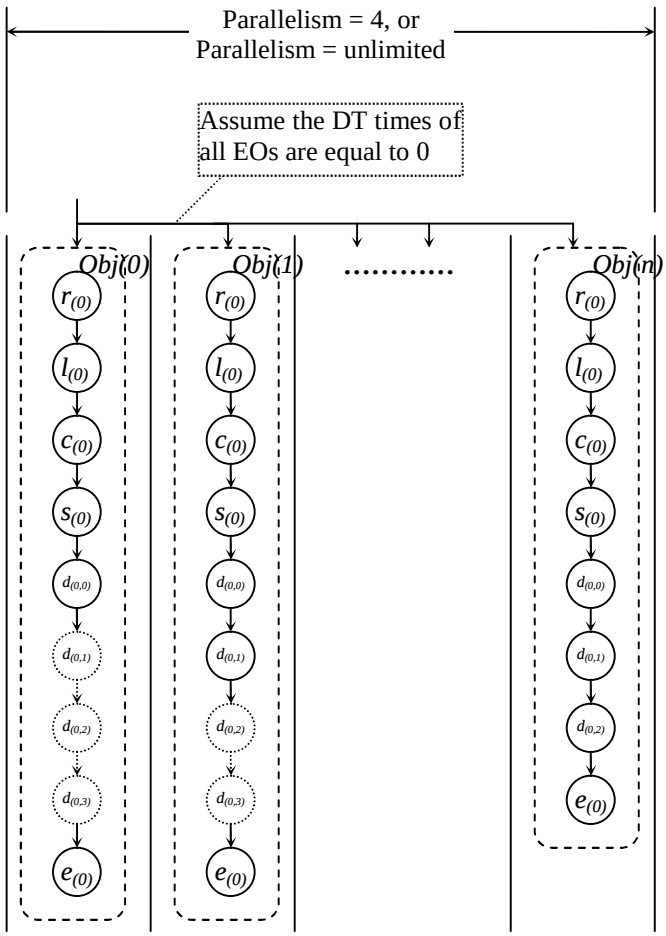


Figure 6.38 Assumptions for the Best Case 1 and Best Case 3



All other assumptions are the same as Best Case 1, 3

Figure 6.39 Assumptions for the Best Case 2 and Best Case 4

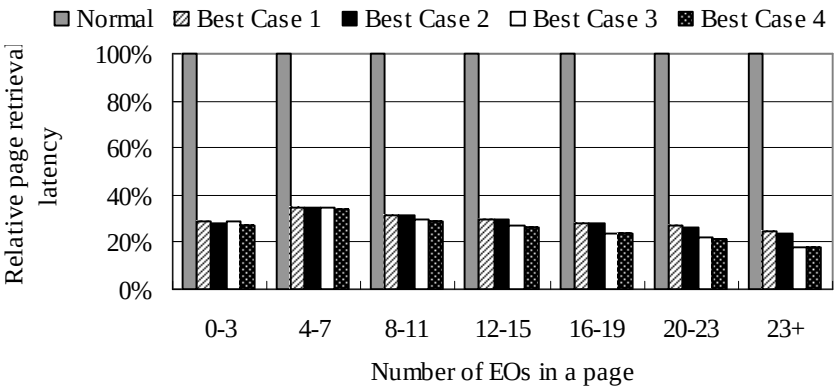


Figure 6.40 Upper bounds of improvement on page retrieval latency

This is not surprising because: Since we assume either all EOs are defined in the first data chunk of CO or the DT times of EOs are zero, so EOs are made known for retrieval very early and they all happen at the same time in a burst mode. When the number of objects in a page is greater than retrieval parallelism, parallelism will

become a bottleneck of web retrieval performance. When parallelism is increased to unlimited, such bottleneck is removed. So the performance improvement for infinite parallelism width is better than those under parallelism of 4.

When the number of EOs in a page is big, wider parallelism width will be more effective. This is because: in such situations, more EOs will be held in waiting state if parallelism is limited. In other words, the demand on parallelism is higher in such situation. Increasing parallelism width would right meet the demand. So, wider parallelism width becomes more effective when the number of EOs in a page is big. That is why we see that the performance difference between “Parallelism=Infinite” and “Parallelism=4” becomes bigger when the number of EOs in a page increases.

6.6 Conclusion

This chapter presented our detailed study and analysis on both object retrieval latency and page retrieval latency based on our WRDM model. The results shed light on the complicated interaction among the factors affecting web retrieval latency, which is largely ignored in previous object-level study in this area. For individual single object retrieval, we see that the CT time and CST time are the two major latency components, and most objects consist of multiple data chunks in their transfer. When objects are put together to form a page, the CT time and CST time are no longer the dominating latency components. Due to the dependency among the objects in a page and limited parallelism width used in web retrieval, the DT time and WT time contribute even more to page retrieval latency than actual object fetch latency does. Our detailed study based on WRDM model reveals the complicated relationship among DT, WT, parallelism and page retrieval latency, and shows that to achieve optimal web retrieval performance would require prudent consideration of all the factors. Simply adjust any one of them will not help much because the performance

will soon be bottlenecked by other factors. Based on the understanding of web retrieval latency using WRDM model, we also analyzed the possible impact of real-time content transformation on web retrieval latency and derived various upper bounds for web acceleration. The abundant and comprehensive results obtained in this study also show that the WRDM model is a very useful and effective tool for studying web retrieval latency.

Chapter 7 Study of Compression in Web Content Delivery

7.1 Introduction

User's perceived latency for web content retrieval is always a big concern to web users and content delivery and distribution network service providers. People want to access information faster for a given network bandwidth. To improve the performance of web content retrieval, caching [9, 10, 11] and prefetching [12, 13, 14] have been introduced. However, the performance of these caching-based mechanisms is limited due to the characteristics of web traffic and the cacheability of web objects [13, 14, 17, 18, 19, 20]. To overcome the limitation, researchers are actively looking into mechanisms which accelerate the downloading process of retrieval objects and pages. Examples of such mechanisms include persistent connection [21, 22], pre-connection [195], parallel fetching [229, 230, 231], bundling [23, 24, 25], delta encoding [26, 27, 28], and compression [28, 206]. These mechanisms are believed to have good potential because they cover a wider range of objects and pages.

In this chapter, we would like to investigate the effect and implication of compression in web content delivery from the detailed chunk level. Here the term “compression” means a mechanism which applies a lossless compression algorithm on textual web objects. The support for such compression has been included in both protocols and web browsers. Protocols have included support for web compression since HTTP/1.0 [36]. HTTP/1.1 further enhanced this support by including more compression algorithms such as gzip, deflate and compress [37]. Most common web browsers such as Microsoft IE and Netscape have also equipped support for web compression since 1998 [206]. So, in current web system, it is possible for web data to be compressed and decompressed with no user interaction at the end point. These actions can even be performed in real-time.

Compression is reported to have good potential in increasing virtual network bandwidth, reducing network traffic and workload on web servers, and reducing download time of web pages [206, 215, 216]. While it is instinctive to understand that it is always going to be faster to transfer a smaller file than a larger one, there are some issues regarding page retrieval latency worth of studying.

Typically, a web page is made up of multiple objects, among which one is called page Container Object (CO) and others called Embedded Objects (EO). CO usually is in the form of an HTML file while EOs are mostly images. An HTML file consists of only ASCII text so it is highly compressible. But images used in web pages are usually pre-compressed and it is difficult to compress them further. So, the CO is often the only object in a page that is suitable for compression. Since CO only occupies part of the total page size, how effective would it be to just apply compression on the CO?

Figure 7.1 shows the distribution of pages with respect to the ratio of CO size vs. total page size. From the graph, we see that for more than 60% of web pages, COs occupy less than 50% of total page size. On average, CO occupies only about 44% of total page size. Therefore, no matter how compressible COs are, the data left for transfer would still be more than half of total page size. Considering that there are other latency components such as location resolution time and connection time which can not be improved by compression, we would expect that the performance improvement that compression can bring in would be much smaller than 50%. So, although COs might be compressed up to many times smaller, the improvement on page latency would not be that significant.

Note that Figure 7.1 also shows that there are about 13.7% of web pages whose COs' size is equal to the whole page size. This is because that these pages consist of only one object which is the CO.

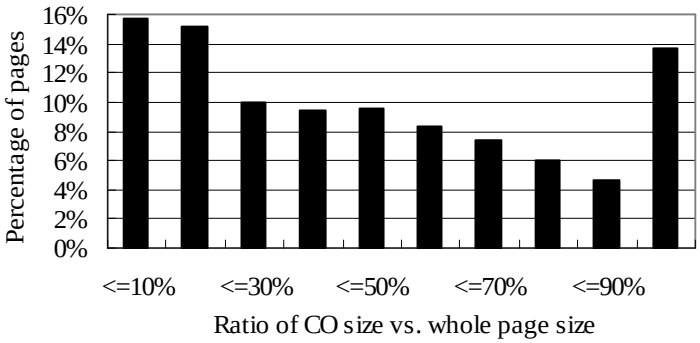


Figure 7.1 Distribution of pages w.r.t. the ratio of "CO size vs. whole page size"

When talking about whole page latency, two critical factors must be taken into consideration: (1) Dependency between CO and EOs of a page, and (2) Parallelism width for simultaneous object fetching. Because EOs are defined in CO, so the retrieval processes of EOs would depend on the retrieval process of CO. When compression is applied to CO and thus affects CO's latency, EOs' retrieval latency would be affected as well and this would in turn affect whole page latency. Parallel fetching of objects in a page is commonly used in most current web browsers. This complicates the relationship between object latency and whole page latency as the overlapping of object latency makes it possible for the fetching of one or more objects to virtually have no effect on the whole page latency. Taking this into consideration, it would be doubtful if compression is still effective when parallel fetching of objects is employed. As compression could have influence on both of the factors, it would be important to study the relationship between them.

There are basically two compression mechanisms in current web system, namely Pre-compression and Real-time compression. Pre-compression compresses objects before they are requested and stores the compressed copies on the server, while real-time compression compresses data on the fly during the actual transmission of the data itself. While people mostly concentrate on the complexity of file management, the real-time feature, and the coverage of static and dynamic objects of these two

compression mechanisms, their performance on whole page latency is not well studied. A detailed study on the difference of the performance of these two compression mechanisms could be useful in helping people to have insight view of them.

The above issues about compression in web content delivery have not been well studied. Little literature regarding these issues can be found. Most of existing studies on compression, e.g. [27, 206, 216] and [191], did not investigate real-time compression, and, none of them studied compression's effect on whole page latency. In this chapter, we report our studies on these issues at the detailed chunk level. By employing fine-grained model and logs, we are able to reveal the complicated relationship between the factors and compression's impact on them. Results show that compression gives lower improvement on whole page latency than it does on single object latency; parallelism width does not affect the effectiveness of compression, in fact, compression is slightly more effective when parallelism width is greater than one; in terms of object latency and page latency, pre-compression always outperforms real-time compression.

In this chapter, we also propose a novel compression mechanism specifically designed for HTML objects to achieve better performance in web content delivery.

The remainder of this chapter is organized as follows. Section 7.2 describes some concepts and background knowledge relative to web compression. Section 7.3 presents our detailed chunk-level study of web compression mechanisms to help us get in-depth understanding on the behavior and performance of them. In Section 7.4, we propose a novel compression algorithm specifically for web content to achieve better performance. Finally, the chapter concludes in Section 7.5.

7.2 Concepts Related to Compression in Web Content Delivery

As we learned in Chapter 4 and Chapter 6, a web page usually is made up of one

Container Object (CO) and multiple Embedded Objects (EO), and the CO is often a basic HTML file and EOs are mostly inlined images. The CO of a page is always the first object returned from the server when the page is requested by client. Both CO and EOs are transferred through network connection from server to client in a streaming way, chunk by chunk.

The latency components for CO and EO are a little different. The retrieval latency for CO is relatively simple. It mainly comes from the retrieval process. But for EOs, the retrieval latency is more complicated due to the dependency between CO and EOs and the limited parallelism width employed by most web browsers for simultaneous fetching of objects. So, in addition to the latency coming from the retrieval process, EOs also have Definition Time (DT) and Waiting Time (WT) in their total retrieval latency (refer to Chapter 6 details). These two latency components play an important role in total page latency. However, they are unnoticed in previous studies on compression.

Web compression is usually achieved by applying a lossless compression algorithm on textual web objects (which usually are the COs). There are basically two ways to apply compression on web objects. The first way is to compress objects beforehand and store the compressed copies on web server to serve future requests. This mechanism is often called Pre-compression. The other way of compression is to compress each chunk of object data on the fly during the actual transmission of the data chunk sequence of the object. This mechanism is referred to as Real-time compression in our study. Considering the streaming nature of web content delivery and dependency between CO and EOs, these two compression mechanisms could have different effect on object latency and whole page latency. In this chapter, we present our detailed chunk-level study on these factors to reveal the effect of these

compression mechanisms on web content delivery. We also propose a new compression mechanism specifically for web content compression to achieve better performance.

7.3 Understanding Compression in Web Content Delivery

7.3.1 Methodology

In our study, we performed retrieval for a large number of web pages in real web environment and obtained detailed chunk-level logs for all compression mechanisms, including the normal situation "No Compression" which is used as a reference in our study.

We first get page URLs from a NLANR trace [276] dated 5th August 2003. Then we replicate those pages content (including EOs) on our web server. We make a pre-compressed version of each page and put it in the same directory as the original page. For real-time compression, we use a reverse proxy to perform the task. We use the zlib compression library [214] to build real-time compression capability into a Squid system [73, 11] version 2.4.STABLE3 to be used as the reverse proxy. Page requests are generated automatically by a web client program pavuk [285]. All requests are forced to pass through a remote proxy in Europe to emulate the real web environment. Detailed chunk-level logs are recorded by the instrumented web client program and forward and reverse proxies. The information about other environment configurations such as software/hardware platform and network environment is described in Chapter 5.

Due to time and space limitation, we stopped the collection of logs when the total log size reached around 17 GBytes. The logs contain a little more than 72,000 web pages, which consists of about 1,010,220 objects. The logs are processed and fed into our simulators to get results for this study.

In the following three subsections, we present the results of our study at chunk-level for insight view of web compression.

7.3.2 General Studies

First of all, we would like to look at some characteristics of web objects and properties of web data transfer that are related to compression.

7.3.2.1 Some Properties about Web Object Transfer

First, let us review some chunk-level results revealed in Chapter 6, which are applicable and important to compressible objects.

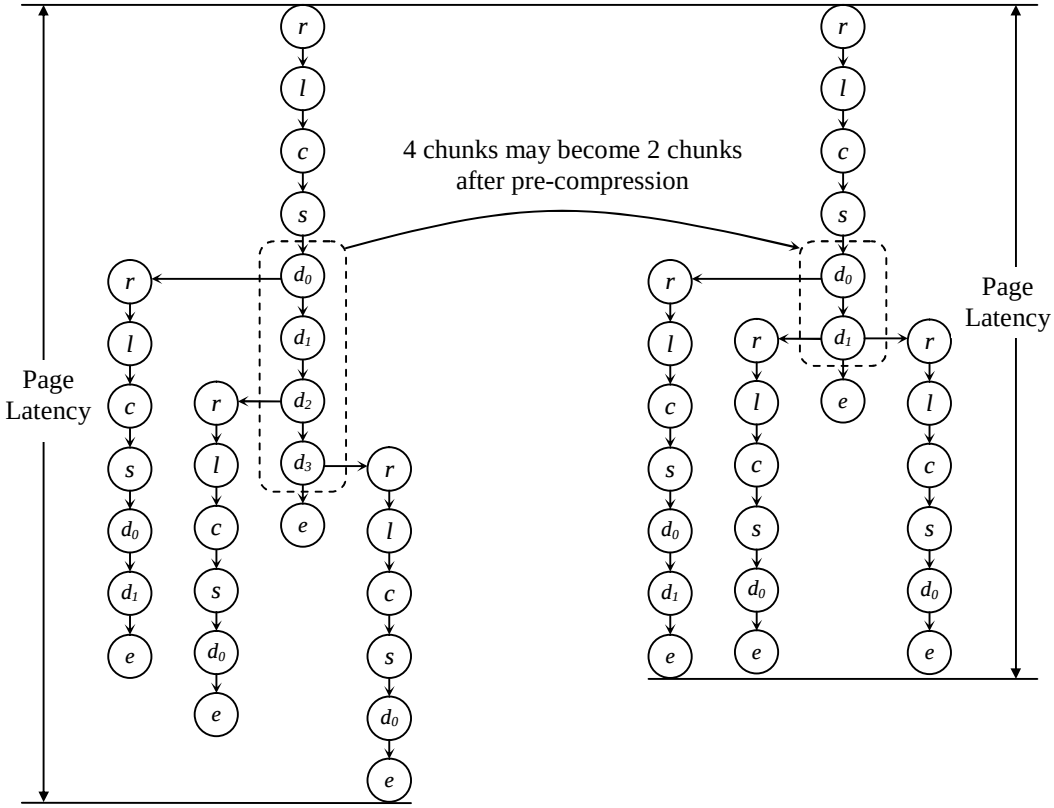
Figure 6.16 shows that a large percentage of COs have sizes between 8 KBytes to 128 KBytes, with an average size of 35.5 KBytes. Since those objects are the candidates for compression, this considerably large-sized distribution gives a good potential for applying compression on them as we know that compression is usually more effective for bigger files.

Figure 6.17 shows that the majority of COs are made up of multiple chunks, and Figure 6.7 shows that the majority (65%) of chunks have sizes between 1 KBytes and 2 KBytes. However, it is worth mentioning that there are also a high percentage of chunks with much bigger sizes above 10 KBytes.

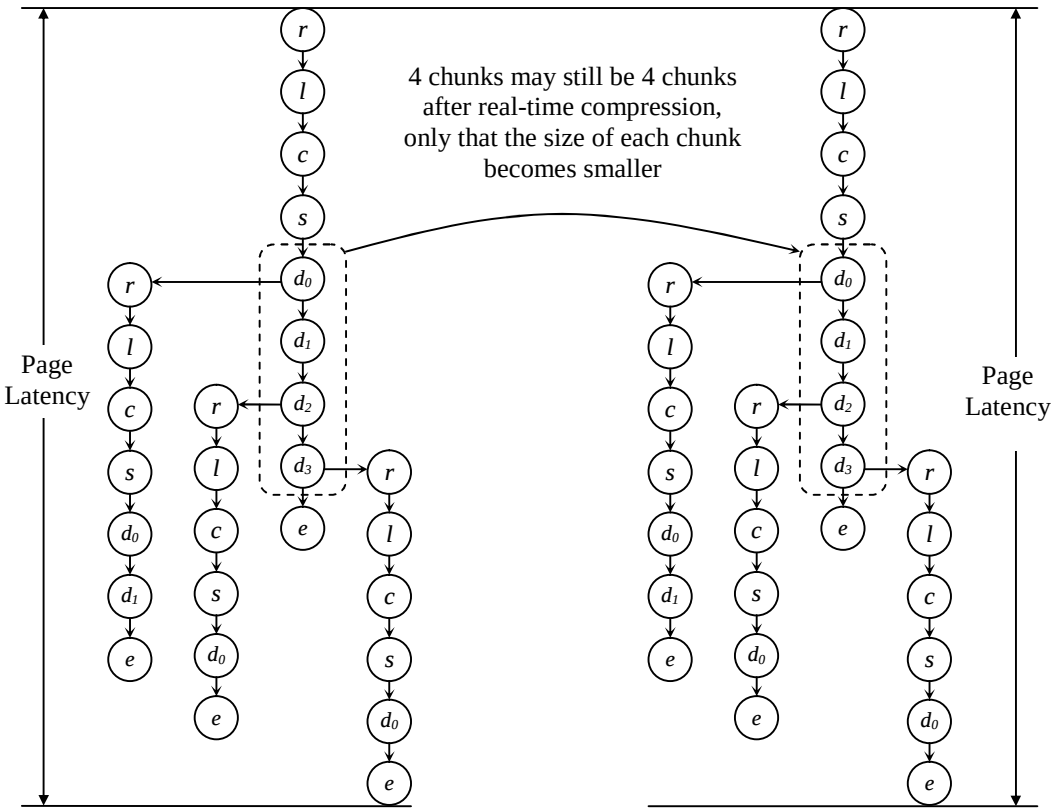
Compression could have influence on both chunk sizes and the number of chunks in an object transfer. Pre-compression reduces object size before the object is requested, so the object data could be delivered by a smaller number of chunks. As for real-time compression, it performs compression in real-time by compressing every chunk in the object transfer. Thus, we would expect that real-time compression would reduce the size of every chunk instead of the number of chunks in the chunk transfer sequence. Since the transfer time of every chunk contributes towards the object retrieval latency, compression would affect object retrieval latency through the influence on chunks.

Since chunk size would be affected by compression, the transfer time for chunks with different sizes would be important to our study. From Figure 6.8, we know that the distribution of the latencies for chunks with different sizes is quite random. The latency for smaller chunks is often comparable to that of much bigger chunks. This observation is important because it indicates that reducing chunk size might not help much in reducing object retrieval latency. We could further deduce that "reducing the number of chunks" might be more effective than "reducing the size of chunks" in terms of reducing object retrieval latency.

Here, we would also like to emphasize the two extreme phenomena revealed in Figure 6.8. One is that the latency for very large chunks is indeed much bigger than that of smaller chunks. This suggests that reducing size for these chunks may still be helpful in reducing their latency. The other phenomenon is that the latency for the "<=1k" group is even bigger than that of "<=2k" to "<=4k" groups. This is mainly due to the TCP slow-start, which indicates that the first a few chunks on a TCP connection are relatively more expensive than the rest. Based on this observation, we would expect that compression might not help much in reducing object latency for small objects. For larger objects with more than 4 chunks in its transfer sequence, the TCP slow-start effect would become less significant as it would be amortized with the transfer of large number of chunks.



(a) Pre-compression



(b) Real-time compression

Figure 7.2 Impact of two compression mechanisms on page retrieval latency

Because compression is mainly applied on COs and there is dependency between CO and EOs, so EOs' latency will also be affect by compression, which would in turn affect whole page latency. Figure 7.2 demonstrates the possible impact of the two different compression mechanisms on page retrieval latency. From it, we suspect that real-time compression may not be as effective as pre-compression in reducing whole page latency.

7.3.2.2 Chunk Level Study on the Effect of Compression on Single Object

In this section, we would like to study the effect of compression on object latency at chunk level.

Figure 7.3 shows relative object latencies with respect to object size for different compression mechanisms. Here, the normal situation "No Compression" is used as the reference. From this graph, we see that both pre-compression and real-time compression have improvements on object latency and the improvement is considerably big. For pre-compression, the improvement ranges from 16.4% to 88.1%, with an average of 57.2%. For real-time compression, the performance gain is from 8.1% to 51.1% and the average gain is 32.3%. The result shows that pre-compression always gives higher improvement than real-time compression does. This could be due to the reason we deduced earlier that pre-compression reduces the number of chunks of an object transfer which is more effective than reducing the size of chunks attained by real-time compression. We will further look into this reason in the later part of this section.

It is a little surprising to see that real-time compression also has big improvement on object latency since it tends to reduce the size of every chunk instead of reducing the number of chunks. Further study shows that real-time compression also reduces the number of chunks in some situation. This is due to a special phenomenon found in

real-time compression. We call this special phenomenon “delay-and-merge” effect and we will discuss it further in the later part of this section.

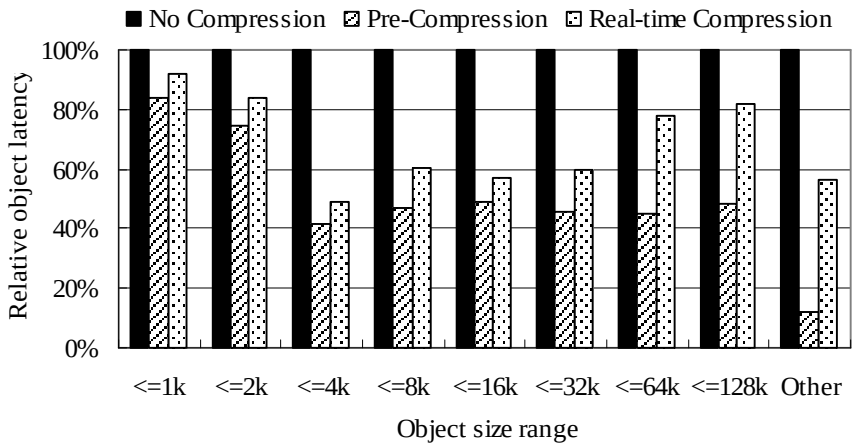


Figure 7.3 Effect of different compression mechanisms on object latency

As object size increases, the performance of pre-compression generally gets better. This could be because of these two reason: first, the compression ratio is normally higher for bigger files; second, when object is small, other latency components (such as connection time) and the TCP slow-start effect are relatively more significant, which makes the effect of compression marginal.

The situation for real-time compression is more complicated. As object size increases, the performance of real-time compression first gets better and then lower, and for the last object size range “Other” (i.e. >128 KBytes), it gets better again. The reasons are related to the “delay-and-merge” effect and we also put the explanations at the later part of this section.

Now, we investigate the reason for compression’s effect on object latency at chunk-level by examining how different compression mechanisms affect chunk sizes and the number of chunks in an object transfer.

Figure 7.4 shows the distribution of chunk sizes under different compression mechanisms. As expected, we see that real-time compression shifts the curve to the left significantly. As many as 78% of chunks are compressed to sizes smaller than 1

KBytes by real-time compression. However, this shifting is ineffective as the latency for 1-KByte chunks is similar to or even higher than that for bigger chunks according to Figure 6.8. On the other hand, we note that real-time compression shifts the 10k+ chunks to smaller chunks. The percentage of chunks belonging to 10k+ group under real-time compression is significantly lower than that of other mechanisms. As we learnt in Figure 6.8, the latency for very large chunks (30k+) is much bigger than that of smaller chunks. So, to compress such chunks would be helpful in reducing the chunk latency.

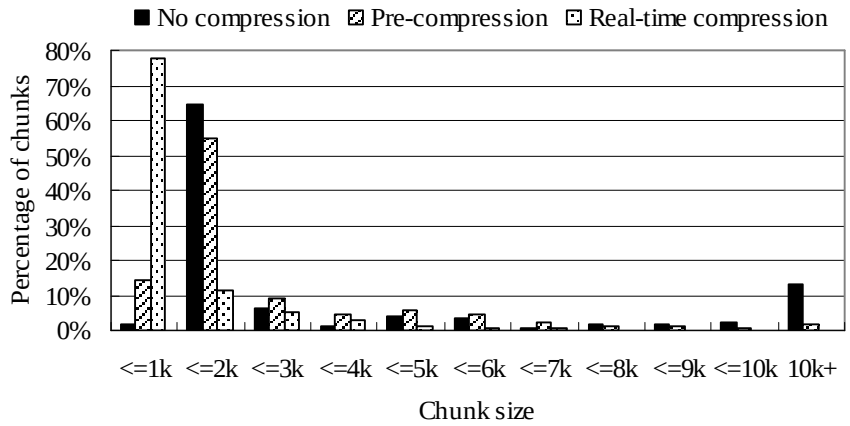


Figure 7.4 Distribution of chunks w.r.t. chunk sizes sent out from server

For pre-compression, the curve is also shifted to the left a little. The reason could be that, after being pre-compressed, more objects become smaller objects and they could be delivered by smaller number of chunks.

Figure 7.5 plots the number of chunks with respect to object size for different compression mechanisms. We see the number of chunks for pre-compression is smaller than normal situation and the difference between pre-compression and normal situation becomes bigger as object size increases. This is actually instinctive to understand because pre-compressed objects are smaller than the original ones so they could be delivered by lesser number of chunks, and, the compression ratio is usually higher for bigger objects, so the difference between pre-compression and normal situation

becomes bigger. With much smaller number of chunks to transfer, it is easy to understand why pre-compression could improve object latency so significantly (see Figure 7.3).

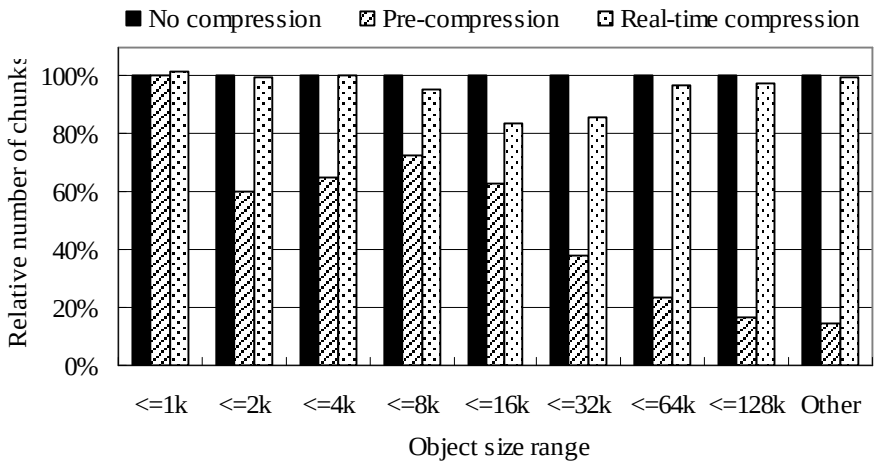


Figure 7.5 Number of chunks w.r.t. object size under different compression mechanisms

It is surprising to see that real-time compression also reduces the number of chunks in some situation since real-time compression is believed to reduce the size of every chunk instead of reducing the number of chunks. Further study revealed a special phenomenon behind. In our experimental system, the real-time compression is performed by a reverse proxy. The reverse proxy receives chunks from the web server next to it and compresses each chunk before sending them out. During the time when the reverse proxy is busy compressing current chunk, the rest of chunks would continuously arrive. Since the reverse proxy is busy, those incoming chunks would be buffered in its buffer and merged into one. Therefore, the size of the following chunk becomes bigger. We name this phenomenon the "delay-and-merge" effect. When object size is big and it has a large number of chunks in its transfer, this effect would accumulate, which would make chunks become bigger and bigger.

Figure 7.6 tries to show this effect by plotting chunk sizes with respect to the chunk sequence number. We see that chunk sizes in real-time compression is generally bigger than that of no compression and pre-compression, and the difference between

them usually gets bigger for chunks with bigger chunk sequence number.

We also note that chunk size becomes bigger for all mechanisms as chunk number increases. This could be also due to the TCP slow-start effect. With successful transmission of more chunks, the transfer rate gets higher (see Figure 6.9) so that a bigger amount of data could be transferred in one chunk.

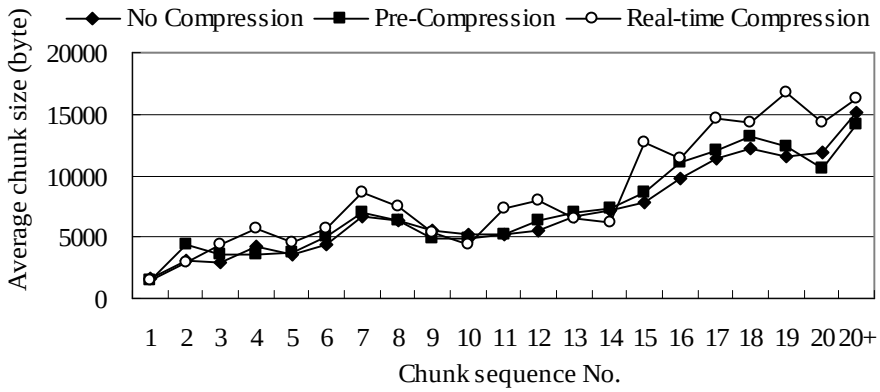


Figure 7.6 Average size of chunks w.r.t. chunk sequence number under different compression mechanisms

The "delay-and-merge" effect has a "warming-up" stage and a "mature" stage. During the "warming-up" stage, chunk size would become bigger and bigger as reverse proxy takes more and more time to compress each growing-bigger chunks. However, because the buffer size in reverse proxy is fixed (64 KBytes in our experimental system), this effect will "mature" when the chunk size grows close the buffer size. In "mature" stage, chunk size would stop growing no matter how many more chunks are still in the transfer sequence.

As chunks would grow bigger due to the "delay-and-merge" effect in real-time compression, the number of chunks for a given object would become smaller than the normal situation. This could explain the result of real-time compression in Figure 7.5. For small objects, real-time compression does not seem to reduce the number of chunks. This is because the number of chunks is too small for the "delay-and-merge" to "warm-up". As object size increases, the "delay-and-merge" effect starts to

“warm-up” so the number of chunks becomes smaller. However, this trend stops when object size is big enough. This is because the “delay-and-merge” effect has matured.

With the above knowledge, we could give explanation on the performance of real-time compression shown in Figure 7.3. Because real-time compression also reduces the number of chunks for an object due to the “delay-and-merge” effect, it is understandable why it also improves object latency. As object size grows from 1 KBytes to 4 KBytes, the performance of real-time compression gets better. This could be because that the “delay-and-merge” effect is in the “warming-up” stage. For objects with sizes between 4 KBytes to 128 KBytes, the “delay-and-merge” effect would get mature so that we see that the performance of real-time compression stops getting better. However, for very big objects with size greater than 128 KBytes, the performance of real-time compression becomes better again. Further study reveals the following reason: for the objects in this group, the chunk size could be very big due to the “delay-and-merge” effect, and very big chunks could be effectively compressed to smaller chunks by real-time compression (see Figure 7.4). With refer to Figure 6.8, the latency for very large chunks (those with size greater than 30k) is much bigger than that of smaller chunks. So, reducing the size of very big chunks would result in reduction in transfer time. Therefore, the performance of real-time compression for this group gets better again.

Considering that most COs consist of 6.7 chunks (see Figure 6.17) and most chunks have sizes between 1 KBytes and 2 KBytes (see Figure 6.7), it would be more effective to reduce the number of chunks than to reduce the size of chunks. This explains why pre-compression always gives higher improvement on object latency than real-time compression does.

In addition, the compression ratio of pre-compression is also slightly better than

real-time compression. Figure 7.7 shows the compression ratio of different compression mechanisms. We see that pre-compression yields compression ratio about 5.2% better than real-time compression does on average. This could be because pre-compression can see the whole object data while real-time compression can only see one chunk of the data. Generally, a compression program which can see the entire input file could compress the file more effectively than the program which sees only part of the input file does.

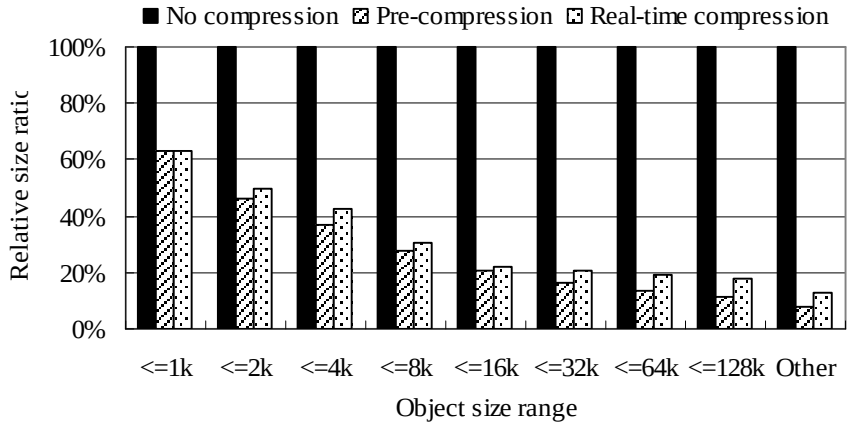


Figure 7.7 Distribution of compression ratio of objects

We also note that the compression ratio of pre-compression becomes higher as object size increases. This is because compression is more effective for big files. While it is easy to understand this, it is not so straightforward to understand the case for real-time compression since it often does not see the entire input file. The reason why real-time compression also generates higher compression ratio with the increasing object size is a little "tricky": When object size is large, the chunks in its transfer sequence tends to be large due to the "delay-and-merge" effect, compression on a single chunk would also be effective when chunk size is large.

Overall, the compression ratio of both of the compression mechanisms is very high. On average, object size can be reduced 87.6% by pre-compression and 82.4% by real-time compression. This further explains the high improvement on object latency

by these two compression mechanisms in Figure 7.3.

7.3.2.3 Effect of Compression on Whole Page Latency

Because the basic unit of browsing is page in current web system, whole page latency is more meaningful to clients than object latency. In this section, we would like to investigate the effect of compression on whole page latency.

As explained earlier, page latency is determined by more complicated factors. So, the improvement on single object retrieval latency achieved by compression may not be translated into the improvement on page retrieval latency directly. Figure 7.8 plots relative page retrieval latency with respect to page sizes. Comparing it with Figure 7.3, we see that the improvement on whole page latency achieved by compression is significantly much lower than it does on object latency. The average performance gain on whole page latency is about 12.2% by pre-compression and 7.4% by real-time compression, as compared to the 57.2% and 32.3% gain on object latency by pre-compression and real-time compression respectively.

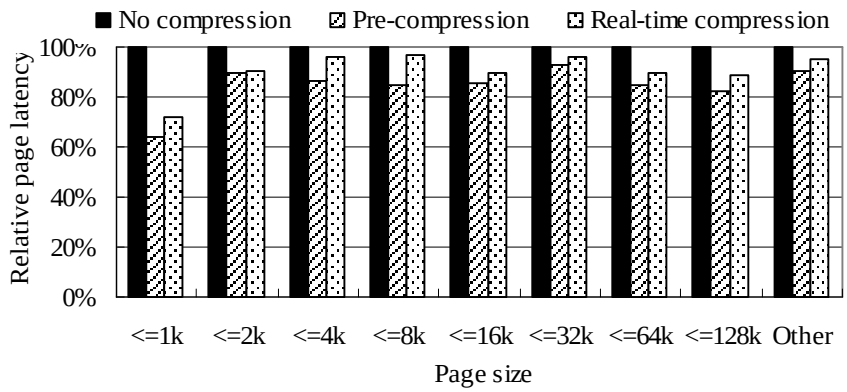


Figure 7.8 Compression's effect on whole page latency (Parallelism = 4)

Although the fact that compressible objects in a page (mainly COs) only occupy part of the total page size could be partially the reason, the big difference between compression's performance on object latency and page latency may also indicate that the two factors, i.e. (1) dependency among CO and EOs of pages and (2) parallelism width for simultaneous object fetching, play an important role in determining page

latency. In the following sections, we would study these factors in detail to get in-depth understanding about compression's effect on whole page latency.

7.3.3 Compression and Dependency

7.3.3.1 Dependency and Definition Time of EOs

We already know that there is dependency between EOs and CO, and such dependency is very importance in determining whole page latency. Here, we first review some of the studies regarding the definitions of EOs in COs.

From the studies in Chapter 6 (Figure 6.12, Figure 6.18, Figure 6.19, Figure 6.20), we see that a considerably large percentage of EOs are defined in the late parts of CO's transfer sequence, and appearing in the chunks with large sequence number. If those definitions could be shift to earlier chunks with smaller sequence number, EOs would be made known to client for fetching significantly earlier. But an observation from Figure 6.20 also indicates that there is limitation on shifting definition points of EOs to earlier parts of CO's transfer sequence because CO undergoes some latency components such as CT time before the actual transfer of data chunks starts.

From Figure 6.19, we see that many EOs are defined in the chunks with large sequence number. If those definitions could be shift to chunks with smaller sequence number, EOs would be made known to client for fetching significantly earlier.

The dependency between EOs and COs causes the extra latency component, i.e. the Definition Time (DT) in the retrieval of EOs. Figure 6.15 confirms that DT is a very important latency component for EOs. So, reducing DT could be an effective way in reducing the retrieval latency for EOs, which could in turn reduce the whole page latency.

7.3.3.2 Compression's Effect on DT of EOs

Since the retrieval of EOs is dependent on CO and compression has significant

influence on CO (see Section 7.3.2.2), the DT times of EOs could be affected by compression. In this section, we present our study of compression's effect on the DT times of EOs at the chunk level.

Figure 7.9 shows the relative DT times of EOs under different compression mechanisms. We see that compression can reduce DT times of EOs considerably. The reduction achieved by pre-compression is higher than real-time compression. This could be due to the reasons studied in Section 7.3.2.2. On average, the DT time can be reduced by 43.6% in pre-compression and 10.7% in real-time compression.

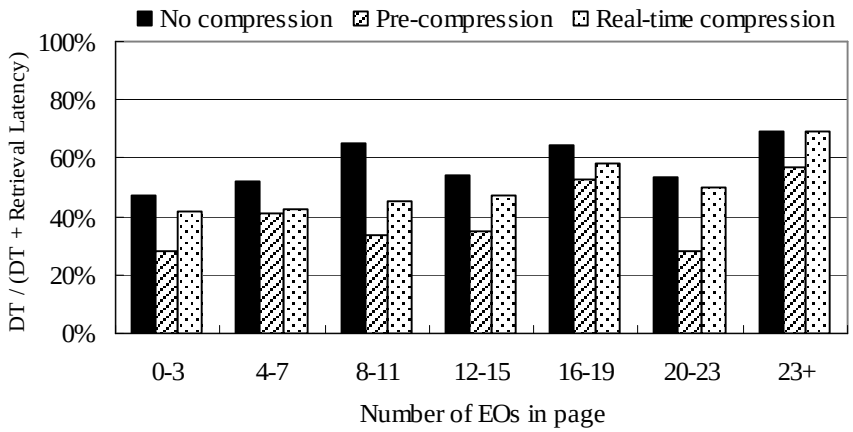


Figure 7.9 Relative DT times under different compression mechanisms

We notice that for the "23+" group in Figure 7.9, the DT time under real-time compression is very close to that of "No Compression". We speculate the reason could be the following: For pages with "23+" EOs, their COs tends to be big in size. When CO's size is big, the "delay-and-merge" effect in real-time compression is more obvious. This effect will make the later chunks bigger and bigger. While the chunks get bigger, they could possibly contain more EOs in them. In other words, more and more EOs are "delayed" to later chunks due to the "delay-and-merge" effect. So, the DT time for the "23+" group gets close to that of "No Compression".

Besides the reasons described in previous sections, Figure 7.10 also shows another reason for why compression reduces DT times of EOs. This graph plots the number of

EOs defined in each chunk for different compression mechanisms. We see that pre-compression shifts the curve to the left significantly. In other words, pre-compression makes more EOs known to client in earlier chunks, which would mean smaller DT for EOs. Real-time compression also shifts the curve to the left, but not that significantly.

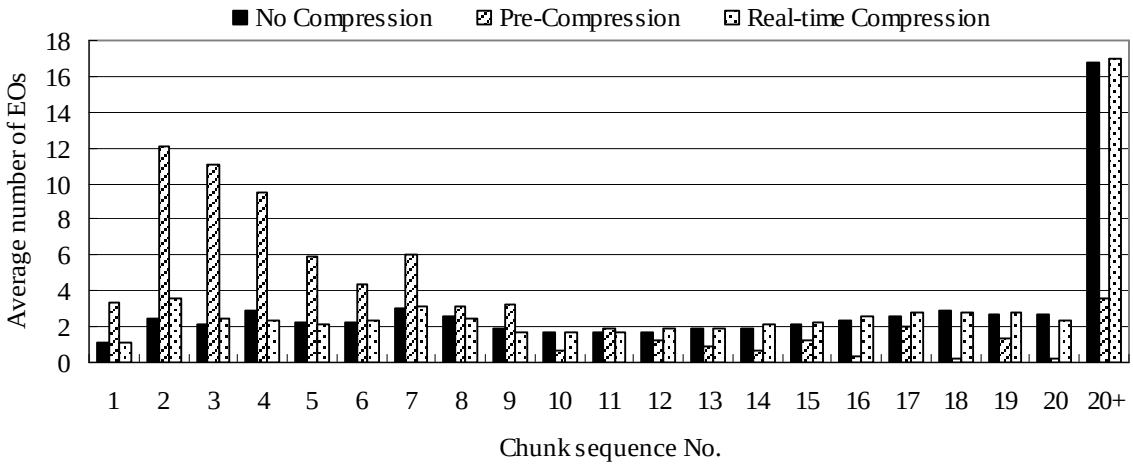


Figure 7.10 Average number of EOs w.r.t. chunk sequence number in CO transfer under different compression mechanisms

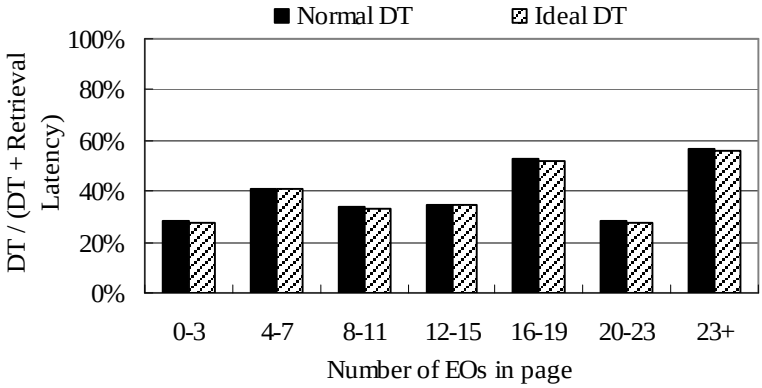


Figure 7.11 Relative values of "DT vs. EO latency" under pre-compression

From Figure 6.9 and Figure 6.20, we learnt that there would be limitation on shifting definition points of EOs to earlier places. Here we would study the upper bound of compression's effect on DT times of EOs. We compute the upper bound by assuming ideal DT for all EOs, i.e. assuming all EOs are defined in the first chunk of CO's transfer. Figure 7.11 and Figure 7.12 show the upper bounds for pre-compression and real-time compression respectively. From them, we see that pre-compression has

reduced the DT times very close to their respective upper bounds, while there is still noteworthy difference between the normal case and ideal case for real-time compression. This indicates that real-time compression is less effective in reducing DT times of EOs.

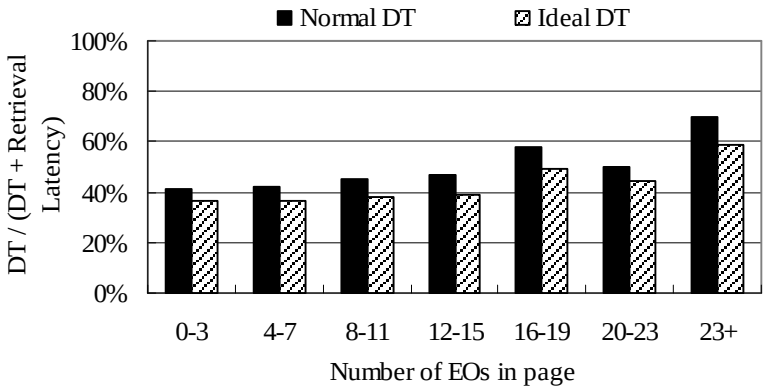


Figure 7.12 Relative values of "DT vs. EO latency" under real-time compression

7.3.3.3 DT and Page Latency

Based on the previous studies, we see that compression could reduce page latency in two aspects. First, compression reduces the size of the CO so that it could be delivered faster (see Figure 7.3). Second, compression would also reduce DT times of EOs since they are dependent on CO's retrieval (see Figure 7.9) (Note that compression would not reduce other latency components for EOs other than the DT time). When the number of EOs in a page is small, the improvement would mainly come from the first effect. But the reduction in DT times of EOs would contribute to the improvement when there are more EOs in the page.

Figure 7.13 shows relative page latency under different compression mechanisms with respect to the number of EOs in a page. We see that the improvement of compression is generally higher when the number of EOs in a page is small. This is because the improvement mainly comes from the reduction in CO's retrieval latency in these cases. For pages with few EOs, the retrieval latency of the CO would be the dominating factor of the whole page latency. Since compression have significant

improvement on CO's retrieval latency (see Figure 7.3), it would also improve page latency considerably for such pages.

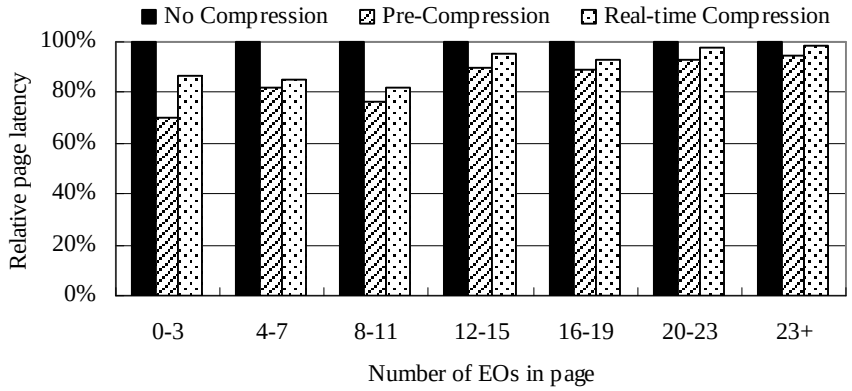


Figure 7.13 Whole page latency w.r.t. number of EOs in a page under different compression mechanisms (Parallelism = 4)

When the number of EOs in a page is big, the improvement achieved by compression is much smaller. This could be due to the following two reasons:

First, when the number of EOs in a page is big, the page latency would be dominated by the of EOs' latency. For EOs' latency components, compression could only reduce the DT times, and the reduction in DT time is not as big as the reduction in CO's retrieval latency (see Figure 7.9 and Figure 7.3).

Second, page latency is also affected by parallelism. When parallel fetching of objects is used, it would be possible for the fetching of one or more objects to virtually have no effect on the whole page latency. In Figure 7.13, we used a parallelism width of four as it is the default value in most current web browsers. This wide parallelism width could dilute the effect of reduced DT times of EOs.

Overall, page latency can be reduced by 12.2% in pre-compression and 7.4% in real-time compression. Compared with the reduction in object latency (see Figure 7.3) and reduction in DT times of EOs (see Figure 7.9), the reduction in whole page latency is much smaller. This could be due to the use of four way parallelism which prevents the reduction in CO's retrieval latency and DT times of EOs from being translated into

reduction in whole page latency directly.

Figure 7.14 and Figure 7.15 further study the upper bound of dependency's effect on whole page latency for pre-compression and real-time compress respectively. The upper bounds are computed by assuming ideal DT for all EOs, i.e., assuming all EOs are defined in the first chunk of CO's transfer.

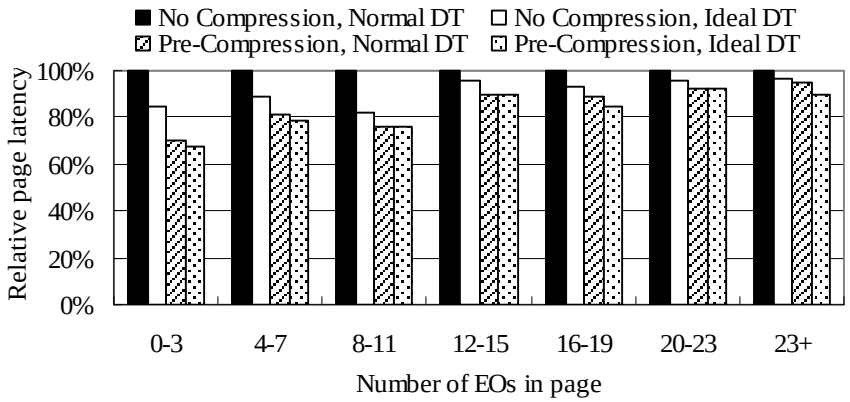


Figure 7.14 Upper bound of dependency's effect on whole page latency for pre-compression

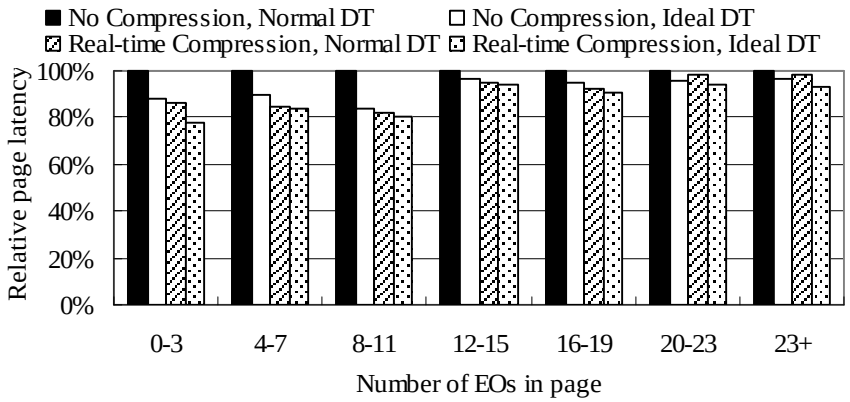


Figure 7.15 Upper bound of dependency's effect on whole page latency for real-time compression

By comparing the situations of "No Compression, Normal DT" and "No Compression, Ideal DT", we see that DT times of EOs do have influence on page latency. This confirms the effectiveness of compression since compression could reduce the DT times of EOs.

It is noteworthy that the performance of "Pre-Compression, Normal DT" is even better than "No Compression, Ideal DT". This could be explained by the following

reasons. Firstly, the difference in DT times between these two situations is very small since pre-compression has reduced DT times significantly (see Figure 7.11). Secondly, pre-compression would also reduce the retrieval latency of COs. Putting these two factors together, it would be understandable that the performance of "Pre-Compression, Normal DT" is even better than "No Compression, Ideal DT".

But for real-time compression, the performance of "Real-time Compression, Normal DT" is sometimes worse than "No Compression, Ideal DT". This is because real-time compression is not so effective as pre-compression in reducing the DT times of EOs and retrieval latency of CO.

Nevertheless, the performance of both pre-compression and real-time compression is very close to their respective upper bounds. This could also be due to the use of parallelism which dilutes the differences of different compressions.

7.3.4 Compression and Parallelism

As parallelism may affect the effectiveness of compression, we would like to further study the performance of compression mechanisms under different parallelism widths in this section.

Current web system utilizes parallelism for simultaneous fetching of objects in a page. Currently, most common web browsers such as Microsoft IE and Netscape use a parallelism width of four for all web page retrievals. However, the effective parallelism width may vary in different environment. For example, in a low-bandwidth environment, the effective parallelism width that clients can enjoy could be as low as one.

Figure 7.16 shows the performance of different compression mechanisms under different parallelism width. While increasing parallelism width would reduce whole page latency for all compression mechanisms, we see that pre-compression constantly

gives the optimal performance in all situations. Also, the performance of real-time compression is always a little better than normal situation. These results are in accordance with the result shown in Figure 7.13. This indicates that a variation in parallelism width does not affect the relative effectiveness of compression.

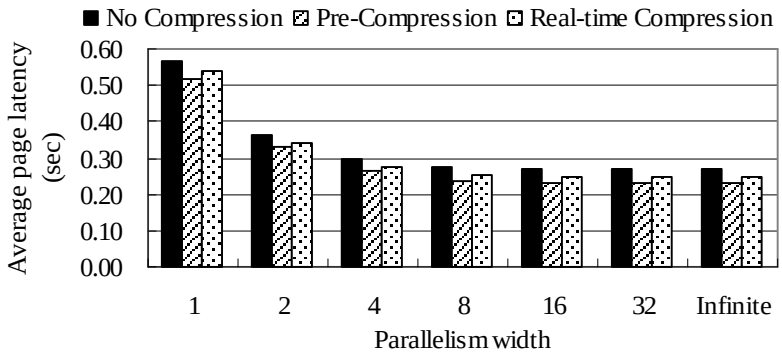


Figure 7.16 Performance of different compression mechanisms under different parallelism width

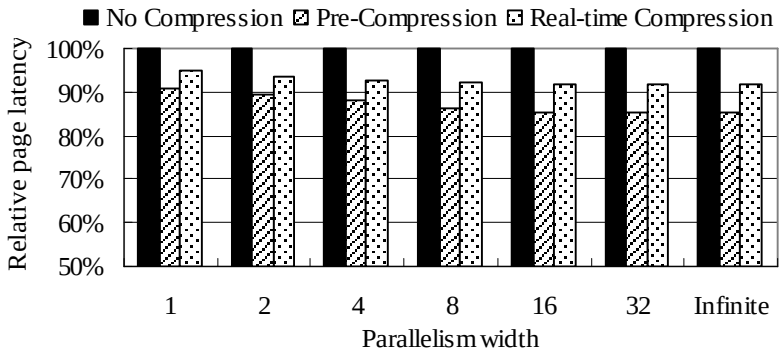


Figure 7.17 Relative performance of different compression mechanisms under different parallelism width

In addition, we also note that the relative improvement of pre-compression actually gets slightly higher as parallelism width increases. This may be observed more obviously in Figure 7.17 which shows the relative performance improvement of different compression mechanisms under different parallelism width. We see that when parallelism width increases from 1 to 32 or greater, the relative improvement of pre-compression increases from 9% to 15%.

The relative higher improvement of pre-compression may indicate that compression is more efficient when parallelism width is big. This could be due to the

higher demand and usage rate that pre-compression imposes on parallelism. Refer back to Figure 7.10, we see that pre-compression shifts significantly large number of definitions of EOs to the earlier chunks of a CO transfer, so more EOs will be made known to client faster and earlier (also refer to Figure 7.9). Thus the demand on parallelism width is higher. This higher demand on parallelism width would result in more EOs being held in waiting state for a given parallelism width. Figure 7.18 shows the percentage of EOs that are held in waiting state under different situations. As expected, pre-compression gives the highest percentage when parallelism width is greater than one (when parallelism width is one, all EOs will virtually have to wait).

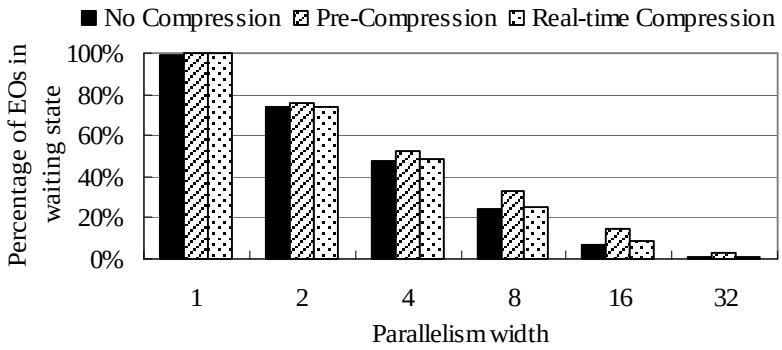


Figure 7.18 Percentage of EOs that are held in waiting state under different parallelism width

On the whole, compression (especially pre-compression) would make a greater number of EOs known for retrieval fast and early and a higher percentage of EOs be held in waiting state for a given parallelism width. In such situation, increasing parallelism width would right meet the higher demand. So, compression becomes more effective when parallelism width increases.

Lastly, we see in Figure 7.16 and Figure 7.17 that the relative improvement of all compression mechanisms becomes insignificant when parallelism width exceeds 8. This could be due to the moderate number of EOs in pages. As shown in Figure 6.12, the web pages in our trace have 13.5 EOs per page on average. When parallelism width grows bigger than 8, there would be very few EOs being held in waiting state. Thus, to

further increase parallelism width would only have trivial effect.

7.4 Content-Aware Global Static Compression for Web Content Delivery

7.4.1 Specific Compression for Web Content

From previous sections, we understand that compression is an effective way in improving web retrieval latency, although the improvement it achieves on page retrieval latency is not as significant as it does on single object latency.

Currently, the compression algorithms employed in web system are general-purpose compression algorithms such as Huffman coding (used by compact), LZW (used by compress) and LZ77-variants like `deflate` (used by zlib and gzip) etc. [286, 211, 287, 212]. In those general-purpose compression algorithms, the content of web objects is treated as a blind byte stream. Those algorithms generally do not provide specific analysis on web-specific content like HTML tags or script language key words. For all the characters and strings, those algorithms just treat them equally and evenly. This method usually works well on common files. But in web content, there may be still room for improvement since there are abundant special strings which may be compressed more effectively if compression algorithms are aware of them and perform special compression on them.

In web objects[3] such as HTML files, certain string tokens like HTML tags are used very frequently. Each of such tokens represents certain special fixed combination of characters. In other words, some characters always go with some others in web content. However, general-purpose compression algorithms do not specially take the advantage of these special fixed combinations of characters. Instead, they just dynamically discover any arbitrary string of characters without considering the special relationship among them. This often results in shorter repeated strings, especially at the

[3] Without explicit emphasis, the term "web object" or "object" in this chapter would refer to "textual web objects". This is because we are talking about lossless compression on textual objects.

beginning phase of compression.

On the other hand, from the global point of view, the special strings in web content occur frequently and repeatedly. However, in a single object, the occurrence of those special strings may not be so frequent and repetitive. So, working on single web objects is generally not as effective as taking the advantage of the globally high frequencies and repetitiveness of the special strings.

Furthermore, the characteristics of the special strings in web content are very stable. For example, they seldom change in terms of spelling, number of strings and global occurrence distribution etc. This allows us to employ global static token-string tables to compress those strings in all web objects, which would be more effective than generating a token-string table for every object and storing the table in each compressed object since the global static tables can be distributed with the compression programs prior to the use of such compression.

From the above discussion, we can see that general-purpose compression algorithms have some effectiveness loss in the specific domain of web content delivery. There still exists some room for improvement.

In this section, we propose a new content-aware global static compression mechanism for web content delivery. This mechanism is specifically designed for web content to take the advantage of the frequently occurred fixed combinations of characters in web content. It can be used as a complementary mechanism on top of existing general-purpose compression algorithms to improve their effectiveness in the specific area of web content delivery.

Note that we are still talking about lossless compression mechanism applied on textual web objects. We assume such textual web objects to be the HTML files in our study, although it may also include other types of files like css, php and asp files etc.

Also, we assume the codes contained in such textual web objects are printable ASCII codes[4], including the carriage return character (0x0D) and the line feed character (0x0A). In terms of ASCII code values, we assume textual web objects only contain ASCII codes between 0 and 127.

7.4.2 Content-Aware Global Static Compression (CAGSC) for Web Content Delivery

7.4.2.1 Introduction

Our basic idea is to compress web objects by replacing the special fixed combination of characters (e.g. HTML tags) found in web object with special short single tokens. We select suitable special strings in web content and pre-generate token-string tables for them, and then we use the token-string tables to compress and decompress the special strings in web objects. This idea is inspired by the algorithms of Huffman and LZW and the frequently occurred special strings in web content. We introduce some new ideas in our mechanism to make it especially effective for the compression on web content.

Firstly, unlike many general-purpose compression algorithms which may encode single characters or arbitrary combination of characters, our mechanism is aware of the special fixed combination of characters in web content. It will look for and treat such combination of characters as inseparable units. In other words, our mechanism is a content-aware compression mechanism specifically for web content.

Secondly, in our mechanism, we pre-acquire a selection of strings for compression based on their global occurrence frequencies. In other words, we select cross-object frequently occurred strings as candidates for compression. The frequencies are pre-computed globally based on a wide-range collection of web objects. Further statistics such as weighted frequencies and potential gains are calculated based

[4] CAGSC compression can also be extended to work on objects containing non-printable ASCII codes by employing techniques like byte-stuffing or using special coding schemes for CAGSC coding.

on the global frequencies of strings for selecting suitable strings for compression.

Thirdly, our mechanism is able to compress multiple types of strings. In web content, there are usually different types of special strings such as HTML tags and JavaScript strings etc. There could even emerge other special strings in web content in the future. Our mechanism can be extended easily to work with arbitrary types of special strings.

Fourthly, the token-string tables for compression are used in a static way. For each type of strings, a token-string table will be generated based on the selected string candidates from that type of strings prior to the actual compression and decompression taking place. Because the characteristics of the strings (e.g. the frequencies and spellings of strings etc.) based on which tables are generated are very stable in web content, so the tables can be treated in a rather static way. Therefore, the essential token-string tables for compression and decompression does not need to be stored and transferred along with every compressed object. This would further improve compression ratio and transfer speed.

Finally, our mechanism recognizes different regions in objects and uses corresponding token-string tables to compress each region. We perform this task dynamically and with a single pass scan of object body. Decompression is carried out in a similar manner, i.e. different token-string tables will be used to decompress different regions of a compressed object.

On the whole, our mechanism works on special strings in web content and compresses them by replacing them with special short single tokens. Because we rely on *global* cross-object frequencies and employ s*tatic* token-string tables in our mechanism, so we refer our mechanism as *Content-Aware Global Static Compression (CAGSC)*. Consequently, we refer to the tokens which are used to replace strings in

CAGSC compression as *CAGSC tokens*.

The CAGSC compression is proposed to only compress web specific strings like HTML tags and JavaScript strings etc.; other parts of the object content will not be compressed by CAGSC. To achieve better compression performance, it is suggested that CAGSC be put to work together with other general-purpose compression algorithms. From another point of view, the CAGSC compression should be regarded as a complementary mechanism to general-purpose compression algorithms in the specific area of web content delivery.

In the situation where CAGSC compression works together with another general-purpose compression algorithm, a web object will first be compressed using CAGSC compression, and then it will be handed over to the general-purpose compression algorithm to do further compression.

Figure 7.19 shows the model of the application of CAGSC compression in web content delivery, and Figure 7.20 gives a simple example of CAGSC compression on an object.

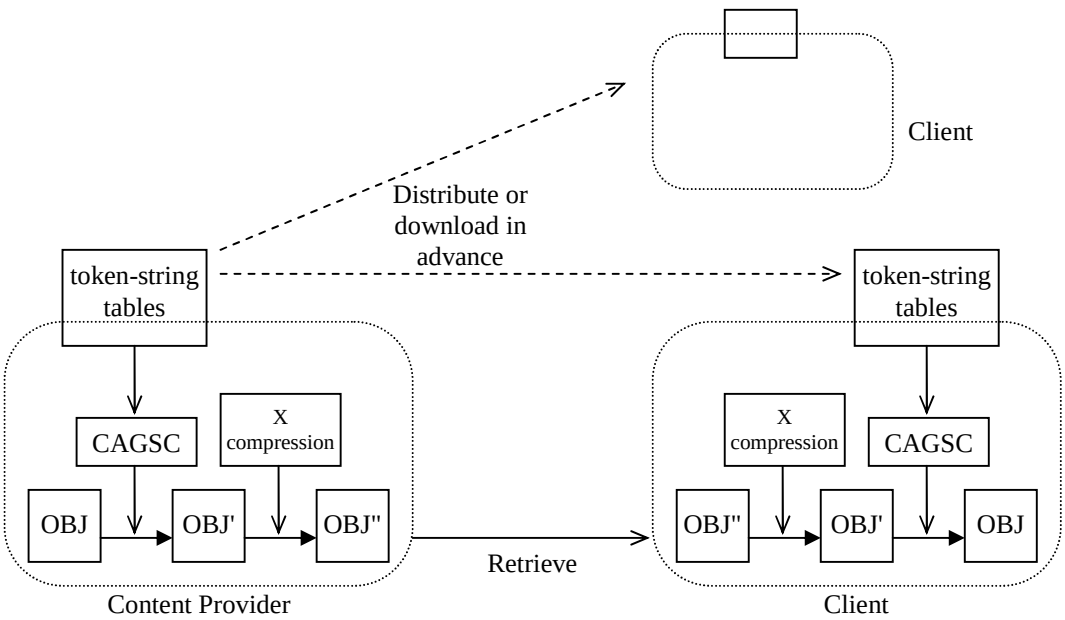


Figure 7.19 Model of application of CAGSC compression in web content delivery

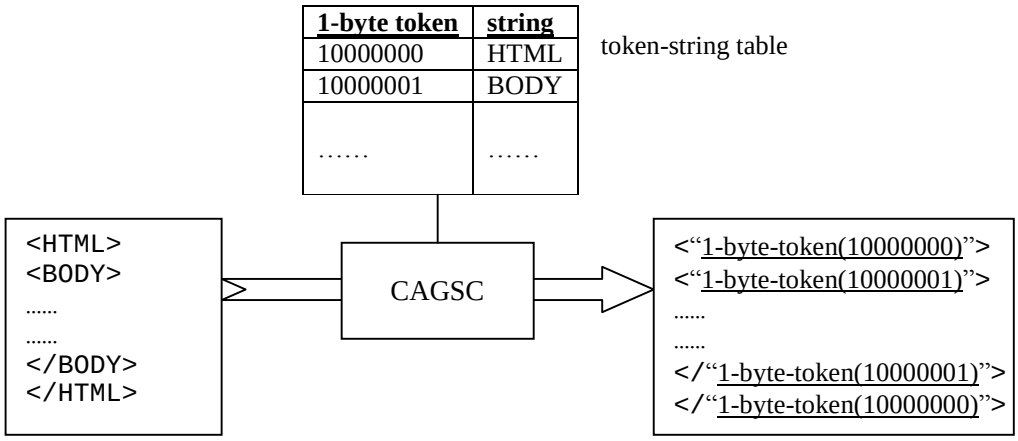


Figure 7.20 Example of CAGSC compression

In the following subsections, we give detailed description of various aspects of CAGSC compression. We first discuss how token-string tables for CAGSC compression are generated (Section 7.4.2.2), then we explain how to apply CAGSC compression in web content delivery (Section 7.4.2.3).

7.4.2.2 Generating Token-String Tables for CAGSC Compression

From the previous section, we see that token-string tables play a key role in CAGSC compression. So we would like to first give the details on how token-string tables are generated.

In brief, the process of generating token-string tables is shown in Figure 7.21. First, a wide-range of web objects are collected. From there, we can get multiple special string sets and the global frequencies of each string in the sets. Then we calculate weighted frequencies and potential gains for the strings and sets (see the following subsection for details). The weighted frequencies and potential gains will be used to decide the coding lengths and what strings to be included in the final token-string tables. Finally, the token-string tables are generated based on the selected strings and coding lengths. These tables will be used for CAGSC compression. , and they are distributed with CAGSC packages to parties prior to the use of CAGSC compression.

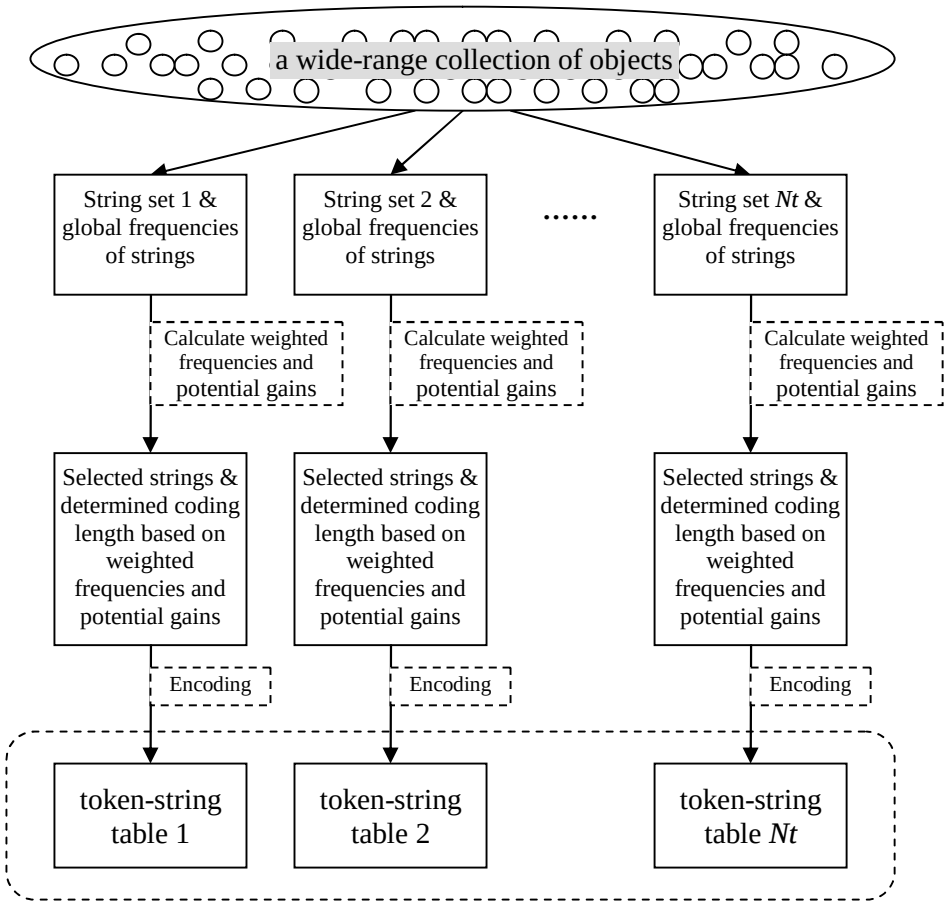


Figure 7.21 Process of generating token-string tables

Below, we describe some details regarding the generation of token-string tables in CAGSC compression.

7.4.2.2.1 Special Strings in Web Content

CAGSC compression works on special strings such as HTML tags in web content. So we would like to first study some properties of such strings.

The following properties of such strings are of interest to CAGSC compression:

Ÿ Different types of strings, i.e. Multiple string sets

The special strings in web content may be of different types, or we can say, there are multiple different types of string sets in web content. For example, these string sets are popular in web content nowadays: HTML tags, JavaScript strings, XML tags etc.

We denote the total number of the different string sets in web content as ***Nt***[5]. Then all the special strings can be represented as the superset S_s of all the different string sets:

$$S_s = \{ S_i \mid S_i \text{ is a string set of a particular type, } 1 \le i \le Nt \}$$

A web object may contain strings from multiple string sets, but the strings from a particular set S_i usually only form a subset of S_i . There may be some strings appearing in multiple string sets, e.g. both HTML and JavaScript have the string "height". However, the semantic meanings of such strings are different in each set. So, the multiple string sets are considered mutually exclusive in terms of semantic meanings.

In CAGSC compression, we compress strings by replacing them with short single tokens. So for every string in a string set S_i, we will need to sign a token to it. The mapping relationship between strings and tokens will be represented as *token-string tables* in CAGSC compression. The tokens for all the strings in a string set make a token-string table for that string set.

To ensure the efficiency of CAGSC compression, it is preferable to include multiple different string sets in the mechanism. There are basically two ways to incorporate multiple string sets in CAGSC. The first method is to merge the multiple string sets S_i into one and treat it as one set, i.e. the superset This method has some deficiencies: firstly, the number of strings in the superset S_s can be very big, which would result in long coding length for CAGSC tokens, and that may not be effective in doing compression; secondly, to merge multiple sets into one superset makes the management of string sets difficult. Whenever the properties of a string set change or there emerges a new string set, the whole superset S_s will be affected.

[5] *Nt* actually means "Number of Tables". Since each string set will have a corresponding token-string table in our mechanism, so the number of string sets will be the same as the number of token-string tables. Therefore, we use *Nt* to stand for the number of token-string tables as well as the number of string sets.

The other method of including multiple different string sets in CAGSC compression is to keep the string sets separate and generate a token-string table for each of them to be used in CAGSC compression. While the second method needs to maintain multiple tables, it provides the flexibility and effectiveness of handling the changes of multiple tables. Also, it allows the reuse of CAGSC tokens among different tables. In our study, we use the second method to handle the problem of multiple string sets.

Ÿ **Number of strings in a string set**

In web content, each string set S_i contains limited number of strings. We denote the total number of the strings in a particular string set S_i as $\boldsymbol{Ns_i}$:

$$Ns_i = | S_i | \quad (\text{ where } S_i \text{ is a string set of a particular type, } 1 \le i \le Nt \text{ })$$

If we use s_j to stand for a string in a string set S_i, then S_i can be represented as:

$$S_i = \{ s_j \mid s_j \text{ is a string, } 1 \le j \le Ns_i \}$$

As we stated earlier, each string set S_i will have a corresponding token-string table in our CAGSC compression mechanism. Note that the number of entries of the token-string table corresponding to the string set S_i may not be equal to Ns_i . This is because, for the strings in a string set S_i , we use certain criteria (which will be discussed in the next subsection) to select some strings from S_i for generating token-string table; it is possible that there are some strings being left out; therefore, the number of entries of the token-string table corresponding to S_i may not be equal to Ns_i .

Ÿ **Frequencies of strings**

The occurrence frequencies of strings may vary from string to string. It is instinctive that to work on strings with high occurrence frequencies would yield good performance.

Given a particular string, it may appear rarely in one object but frequently in

other objects, or vice versa. If we only look at one or a few objects, we may get biased occurrence frequencies of strings, which would result in biased performance distribution. Our CAGSC compression aims to achieve good balanced global performance. So we need to obtain the occurrence frequencies of strings that are applicable globally. To achieve this goal, we collect a wide range of web objects, and do an analysis of the cross-object accumulative statistics of strings to obtain their global occurrence frequencies. Works based on such global occurrence frequencies would lead to good balanced global results since such frequencies are independent from any particular objects and therefore applicable globally.

In our study, we denote the global occurrence frequency of a string s_j as $\boldsymbol{f_{sj}}$.

Ÿ **Lengths of strings**

Each string is of certain length, i.e. number of characters (bytes) contained in the string. String length is also an important factor that we need to take into consideration when we do CAGSC compression. For example, if the length of string is even shorter than the CAGSC tokens, then there would be no gain (in fact, there is a loss) to replace that string with a CAGSC token.

In our study, we denote the length of a string s_j as $\boldsymbol{l_{sj}}$.

7.4.2.2.2 CAGSC Coding for Strings

CAGSC compression compresses the special strings in web content by replacing them with short CAGSC tokens. Now, let us look at some issues regarding CAGSC coding.

Ÿ **Fixed-length coding vs. variable-length coding**

There are basically two coding schemes for generating CAGSC tokens. The first type is fixed-length code like the one used in LZW compression algorithm. The other type is variable-length code such as Huffman coding [288]. In our CAGSC

compression, we choose to use the fixed-length code because of the following reason:

As we stated earlier, CAGSC is a complementary mechanism to other general-purpose compression algorithms and should often be put to work together with them to achieve better overall compression performance. After an object is first compressed by CAGSC algorithm, it will be handed over to the general-purpose compression algorithm to do further compression.

Most general-purpose compression mechanisms rely on discovering repeated byte sequence patterns in data to achieve the goal of compression. Examples include those LZ77-based compression mechanisms such as zlib, gzip, zip and pkzip etc. Such compression mechanisms compress data by keeping track of the last N bytes of byte sequence and replacing repeated pattern of byte sequence with a pair of values corresponding to the position of the pattern in the previous data and the length of the pattern.

Because the lengths of variable-length codes may not be of integer times of byte-length, i.e. variable-length codes may not end at byte boundary, so we will need special mechanism (like Huffman coding) to keep track of the boundary of variable-length codes. This would often incur considerable overhead. On the other hand, even if there is such a special mechanism to determine the boundary of variable-length codes, other general-purpose compression mechanisms will not know it. Instead, they will just treat the CAGSC-compressed object as normal byte-sequence data and still cut off the data at normal byte boundary. Because of this, we have found that the use of variable-length codes to first compress certain strings in the data would greatly destroy the repeated byte sequence patterns in the data for byte-boundary-cutting-off compression mechanisms. This significantly reduces the effectiveness of the general-purpose compression algorithm that comes after CAGSC

compression.

However, if we use fixed multi-byte-length-bounded code in CAGSC compression, the repeated byte sequence patterns in the data will be preserved after the data have been first compressed by CAGSC compression. Therefore, when another general-purpose compression mechanism is later applied on the data, it could still achieve its effectiveness in compressing the data.

So, in our CAGSC compression, we employ the fixed multi-byte-length-bounded code, and we make the length of the codes to be of integer times of byte-length, e.g. 1-byte or 2-byte etc.

Ÿ Coding scheme: coding length & coding space

As we just stated above, in CAGSC compression, we make the length of the codes to be of integer times of byte-length, e.g. 1-byte or 2-byte etc. Different coding lengths have different coding space, i.e. the number of tokens that a certain coding length can give. Here we study the issues regarding coding length in CAGSC compression.

An object compressed by CAGSC compression will contain a mixture of CAGSC tokens and other codes of uncompressed data. We need a mechanism to differentiate the CAGSC tokens from other codes.

Because we assume textual web objects contain ASCII codes between 0 and 127 (see Section 7.4.1), so for 1-byte CAGSC tokens, they can only use the values from 128 to 255. That is, the space of 1-byte CAGSC coding is 128 tokens.

Now let us look at the situation of n-byte coding, where $n \geq 1$. Given a coding length n, we will always be able to differentiate a CAGSC token from normal ASCII codes as long as we can differentiate the first byte of the CAGSC token. This is because, once we differentiate the first byte of a CAGSC token from normal ASCII

codes, then we can just read the following *n*-1 bytes and put them together with the first byte to form the CAGSC token. This indicates that we do not need to differentiate the following *n*-1 bytes of a CAGSC token from normal ASCII codes. Therefore, each byte in the following *n*-1 bytes can use values from 0 to 255. Figure 7.22 illustrates this situation.

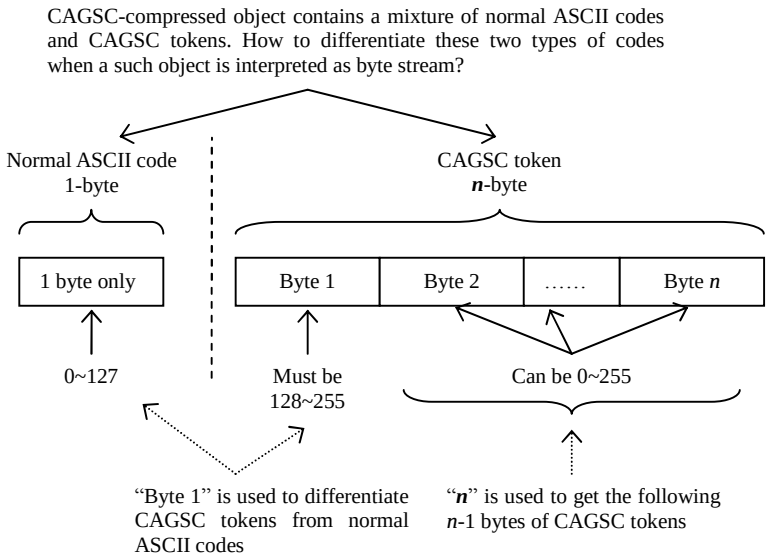


Figure 7.22 *n*-byte coding scheme for CAGSC compression

From the above description, we can deduce that the coding space of *n*-byte coding for CAGSC compression is given by the following formula:

$$CS(n) = 128 \times 256^{(n-1)} = 256^n/2 \qquad (\text{where } n \geq 1) \qquad \text{(F7.1)}$$

In the rest part of this chapter, we denote the coding length of a particular CAGSC coding scheme as l_c. So the coding space for l_c would be:

$$CS(l_c) = \quad 256^{lc}/2 \qquad (\text{where } l_c \geq 1) \qquad \text{(F7.2)}$$

From formula F7.2, we see that coding space, i.e. the number of tokens that CAGSC coding scheme can give, is determined by the length of coding, and the relationship is exponential. Therefore, we expect to see that coding space would grow very quickly as the coding length increases. Table 7.1 lists the number of tokens for a few coding lengths.

In CAGSC compression, we use one coding length for one string set. That is, all the strings within the same string set will be assigned CAGSC tokens of the same coding length. If the coding space of that particular coding length is not enough to cover all the strings in the string set, then we need either to increase coding length or to remove some strings from the set. To increase coding length is simple and it would always be able to solve the problem. However, it may affect the performance of CAGSC compression since the tokens used to replace strings are longer. As to removing some strings from a string set, there are complicated factors affecting the selection of strings. In the next subsection, we are going to study this issue.

Table 7.1 Coding space for some coding lengths

Coding Length	Coding Space
1-byte	128
2-byte	32,768
3-byte	8,388,608
…	…
l_c-byte	$256^{lc}/2$

7.4.2.2.3 Weighted Frequencies and Potential Gains of Strings

There are situations in which we want to select only part of the strings from a string set (i.e. a subset of the whole string set) to be included in CAGSC compression. Such situations include that we want to use a coding space which is smaller than the number of strings in the string set, or we want to keep token-string tables small so that they can be transferred dynamically and quickly, and so on.

Given a string set S_i , the total number of strings it contains is Ns_i . Suppose we have a number Ns_i', where $1 \leq Ns_i' \leq Ns_i$, there is a issue about how to select Ns_i' strings from S_i so that the resulting performance of CAGSC compression on S_i is the best. This question involves the factors of string frequencies, string lengths and coding lengths etc. Below we introduce two parameters to help answer this question.

Ÿ Weighted frequencies of strings

An instinctive way of selecting a good subset of strings from a string set is to

select those strings which have high occurrence frequencies. However, because strings are of variable lengths and the CAGSC tokens used for encoding a string set is of fixed length, so it could happen that the lengths of some strings are longer than the token length while some others may be shorter. Of course it is only meaningful to replace a string whose length is longer than the token length. Taking this factor into consideration, we introduce a parameter named weight frequency of a string to help in the selection of good strings.

For a given string s_j and coding length l_c, the weighted frequency for s_j , denoted by f_{wsj} , is defined by the following formula:

$$f_{wsj} = f_{sj} \times (l_{sj} - l_c) \qquad (\text{where } l_c \geq 1) \qquad \text{(F7.3)}$$

The actual meaning of the weighted frequency of a string is the potential gain from including that particular string in CAGSC compression under certain coding length.

The weighted frequencies of strings can be used for selecting strings to be included in CAGSC compression so as to achieve the best performance for the given coding length l_c and the given number of strings Ns_i' to be selected. Essentially, this can be done by sorting strings according to descendent order of their weighted frequencies, and then select the top Ns_i' strings. When selecting, only those strings whose weighted frequencies f_{wsj} , are greater than zero should be selected. This is because, to include strings with weighted frequencies equal to or less than zero would not give you any performance gain; in fact, it could even cause performance loss.

Note that there is a limitation in this rule: The given number of strings Ns_i' must be smaller than or equal to the coding space of l_c , i.e. $Ns_i' \leq CS(l_c)$. This is because that weighted frequencies are related to coding length. If $Ns_i' \geq CS(l_c)$ so that l_c has to be increased, then the weighted frequencies for all strings would change also, which

would affect the final decision. To handle the situation where l_c may be changed (as well as Ns_i'), we introduce another parameter named potential gain of a subset of strings to solve the problem.

Ÿ Potential gain of a subset of strings

There is a tie between the number of strings Ns_i' to be selected from a string set S_i and the coding length l_c used for encoding the strings. When $Ns_i' \geq CS(l_c)$, l_c has to be increased so that the new $CS(l_c') \geq Ns_i'$. On the other hand, if Ns_i' very small such that it can be covered by the coding space of a smaller l_c , then we can reduce l_c to a smaller value.

When there is a change on Ns_i' or l_c , we need to re-select strings to ensure the best performance of CAGSC compression for the new Ns_i' or l_c . We introduce the potential gain parameter of a subset of strings to help with such evaluation process.

For a given a string set S_i , S_i' is a subset of S_i , then the potential gain of S_i' , denoted by $\boldsymbol{PG(S_i')}$, is defined by the following formula:

$$PG(S_i') = \sum_{j=1}^{Ns_i'} f_{sj} \times (l_{sj} - l_c) \quad \text{(where } s_j \in S_i', Ns_i' = |S_i'|, l_c \geq 1 , Ns_i' \leq CS(l_c)) \quad \text{(F7.4)}$$

The actual meaning of $PG(S_i')$ is the overall potential gain from including the all the strings in the subset S_i' in CAGSC compression.

The potential gains of subsets and the weighted frequencies of strings together can be used to determined the appropriate Ns_i' and l_c for a given string set S_i and select the appropriate Ns_i' strings that would result in the best performance of CAGSC compression. This is usually carried out in the following way:

If Ns_i' and l_c are fixed, then we use the above described method to select the appropriate strings, i.e. we sort strings according to descendent order of the calculated weighted frequencies, and then select the top Ns_i' strings.

If Ns_i' and l_c are not fixed, we set l_c = 1..n, Ns_i' = min($CS(l_c)$, Ns_i). For each pair of Ns_i' and l_c , we use the above method to select the top Ns_i' strings, and then we calculate the potential gains each pair of l_c and Ns_i' , i.e. $PG(S_i')$. Among all the $PG(S_i')$, we select the highest one. Then the pair of Ns_i' and l_c corresponding to the highest the $PG(S_i')$ would be the final decision.

Note that, besides the above parameters, there are other considerations which may affect the number of strings (i.e. Ns_i') to be selected from string set S_i . For example, in the situation where tables need to be transferred dynamically and the network bandwidth is narrow, smaller sized tables may be preferred so that the transfer latency can be minimized.

For each string set in web content, we use the above methods to select the strings from each set and determine the coding length for each set. Then we generate token-string tables to be used in CAGSC compression. Below we discuss the format and other issues regarding token-string tables.

7.4.2.2.4 Token-String Tables in CAGSC Compression

Ÿ Multiple token-string tables & Scalability of CAGSC compression

There are multiple different types of string sets in web content. As we stated earlier, it is preferable to include multiple string sets in CAGSC compression, and we generate and maintain a separate token-string table for each string set. The token-string table corresponding to the string set S_i is referred to as T_i . The total number of different token-string tables is represented as Nt .

It is possible that new string sets may be introduced into web content in the future, or an existing string set has got some changes. It is highly desired that CAGSC compression can be scalable to any new string sets.

To enable CAGSC compression to be scalable to new string sets, as well as to

distinguish different multiple token-string tables, we assign a unique ID to each table. The ID's can be of any types of values as long as each one is unique; but in this study, we define such IDs to be unique integer numbers. We use ID_i to denote the unique ID assigned to table T_i .

When the tokens from a particular token-string table T_i is used, its ID ID_i will be used together with the tokens (see next subsection for details on this), so we can distinguish which table the tokens belong to. This also allows us to reuse the same tokens among different tables. Furthermore, whenever a new string set is introduced into web content, we can include it into CAGSC compression by generating a token-string table for it and assigning a new unique ID to the table. This way, CAGSC compression would be able to be extended to any new string sets in the future.

- **Size of token-string table for a string set**

The size (i.e. the number of entries) of a token-string table for a string set S_i is determined by the number of strings Ns_i' selected from S_i to be used in CAGSC compression. In the previous subsections, we have studied various aspects and introduced parameters to help with the selection of strings from a string set. When the selection is done, the number of the strings becomes fixed; accordingly, the size of the token-string table for the selected strings will be known.

- **Format of token-string tables**

Given a string set S_i , once the selection of strings from S_i is finished and a coding length is determined, a token-string table will be generated for the selection of strings.

The format of token-string tables is given in Figure 7.23. All token-string tables are made of two parts: description part and coding part. In the description part, the unique ID assigned to the table and the coding length used by the table are presented,

followed by an <end tag> (e.g. we may reserve the code "128" to be used as the <end tag>). The coding part contains the "token, string" pairs. Every string in the selection of strings has a corresponding "token, string" pair, and each "token, string" pair is ended with an <end tag>. The token for each string is unique, and all the tokens appeared in the same table are of the same coding length. As to the details on the tokens, please refer to the "coding scheme" part in Section 7.4.2.2.2.

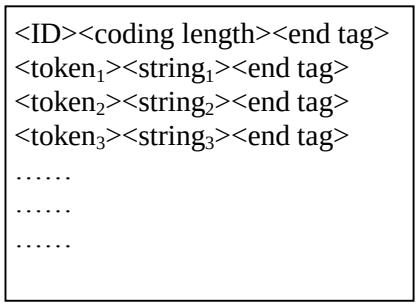


Figure 7.23 Format of token-string tables

Note that every token-string table in CAGSC compression has a unique ID. The ID of a table will be used together with the tokens in the table (see next subsection for details on this). This mechanism would enable us to distinguish tokens from different tables, to reuse the same tokens among different tables, as well as to include new tables into CAGSC compression.

Ÿ Using token-string tables

The token-string tables in CAGSC compression are generated for special string sets such as HTML tags and JavaScript strings etc. The characteristics of such string sets in web content are very stable. For example, they seldom change in terms of spelling, number of strings in a set and global occurrence frequencies etc. As we described in the previous subsection, our CAGSC compression selects strings based on analysis of a wide-range collection of objects. Because of these reasons, the pre-generated token-string tables based on the selections of strings are applicable globally to all objects, and they can be treated rather statically. They do not need to be

stored in and dynamically transferred with every compressed object. Instead, we can distribute them together with the CAGSC compression and decompression packages prior to the use of CAGSC compression.

On the other hand, dynamic download of those tables is also supported in CAGSC compression. This is to handle the rare situations when the tables are updated or a client happens to lack of certain tables. When a client encounters such situations when it decompresses an object, it will request and download relative tables dynamically from the content provider and then go on with the decompression.

7.4.2.3 Applying CAGSC Compression in Web Content Delivery

7.4.2.3.1 Compression Process

Basically, our CAGSC compression compress web objects by scanning object body and replacing the special strings with CAGSC tokens.

To differentiate the CAGSC tokens from the normal ASCII codes, as well as to differentiate the CAGSC tokens from different token-string tables, we introduce two new tags to specify the working areas of CAGSC compression and the ID's of the token-string tables used in each working area. The tags and their formats are as follows:

Starting tag : `<CAGSC Table=ID>`

Ending tag : `</CAGSC>`

The starting tag is used to mark the starting point of a working area of CAGSC compression in the body of an object, while the ending tag is used to mark the ending point of a CAGSC compressed area. The starting tag and ending tag must appear in pair when applying CAGSC compression. Between these two tags, only normal ASCII codes and the tokens from the token-string table with the specified `ID` can appear.

Figure 7.24 illustrates the compression process of CAGSC compression. The

CAGSC compression program scans the body of an object, when it recognizes a special string belonging to a particular token-string table T_i , it will insert a starting tag "<CAGSC Table=ID_i>", and then replace the string with a CAGSC token from table T_i . Following that, if it keeps meeting other strings that belong to the same table T_i [6], it will just replace them with tokens without inserting new starting tags. If the CAGSC compression program recognizes a special string that belongs to another table T_j , it will end the working area for table T_i by inserting an ending tag "</CAGSC>", then it starts a new working area for table T_j . This process repeats until the end of the object body. In Figure 7.24, we suppose the strings str_{ix} belong to table T_i and strings str_{jx} belong to table T_j . We see that CAGSC finishes the working area for table T_i and starting the working area for table T_j when it meets the string str_{j1} .

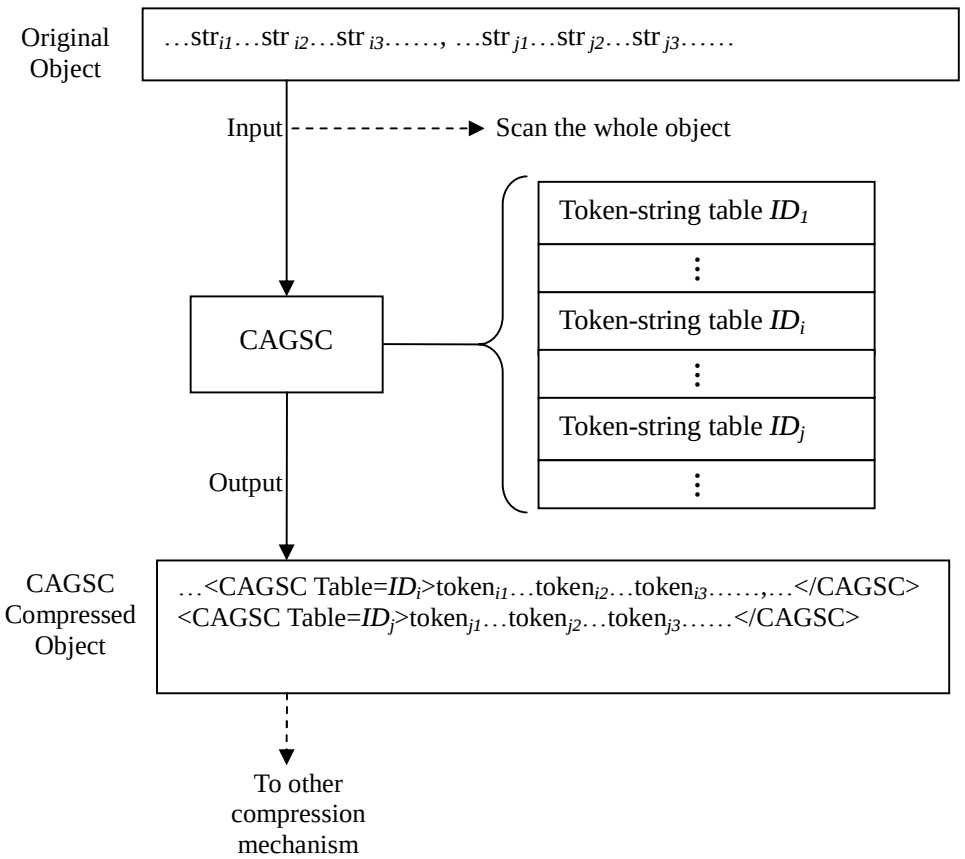


Figure 7.24 Compression process of CAGSC Compression

[6] In web content, this is often the case. For example, we usually meet plenty of JavaScript elements at the beginning part of an object, and many HTML tags at the rest parts.

When the above process finishes, we will get a resulting file which is compressed by CAGSC compression. Normally, this file would contain multiple "<CAGSC Table=ID_i>" and "</CAGSC>" tag pairs. These pairs split the object body into multiple regions; each region is compressed by certain token-string table, and the tables for each region are independent of each other. The CAGSC tokens can be reused among different regions (in fact, they are reused among different token-string tables). Figure 7.25 gives an example of CAGSC compression with two tables. It shows how object body is compressed into multiple regions and how the CAGSC tokens are reused among the regions.

After an object has been compressed by CAGSC compression program, it may be passed to other compression mechanisms to compress further.

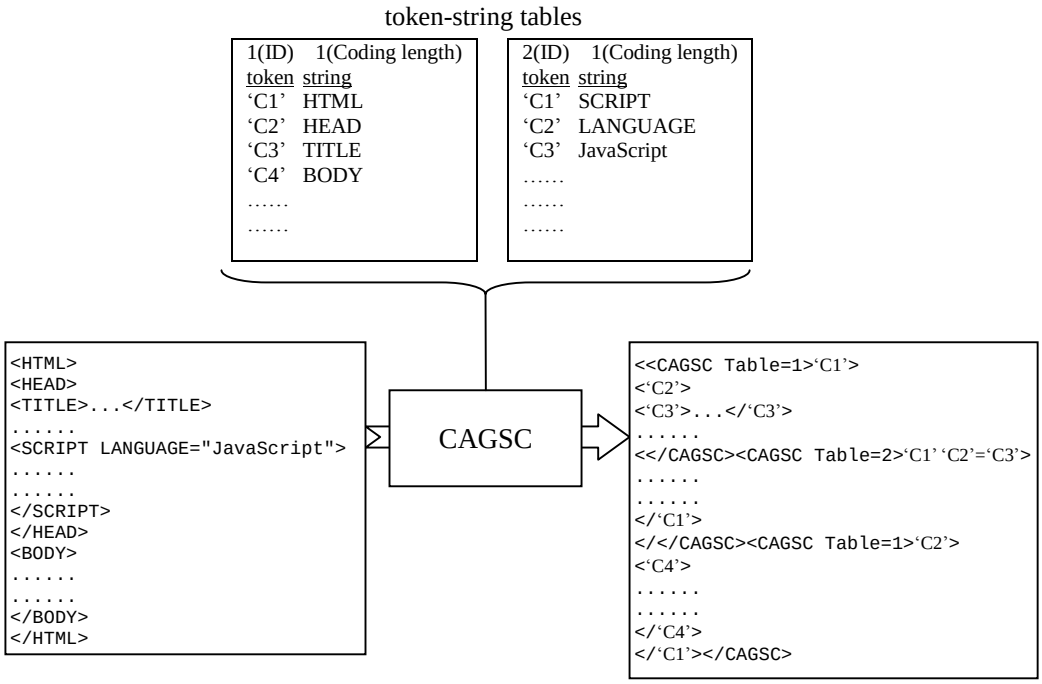


Figure 7.25 Example of CAGSC compression with two tables

7.4.2.3.2 Decompression Process

The decompression process of CAGSC is quite simple. It is just the reverse process of the compression process. Figure 7.26 illustrates the decompression process of CAGSC compression.

The CAGSC decompression program scans the body of a CAGSC-compressed object, when it meets a starting tag "<CAGSC Table=ID_i>", it will use the strings from the token-string table T_i to replace the following met CAGSC tokens. This replacement stops taking place when it meets an ending tag "</CAGSC>". When a new starting tag is met, the program will switch to the corresponding token-string table and start to replace CAGSC tokens with strings. The above process continues until the end of the object body.

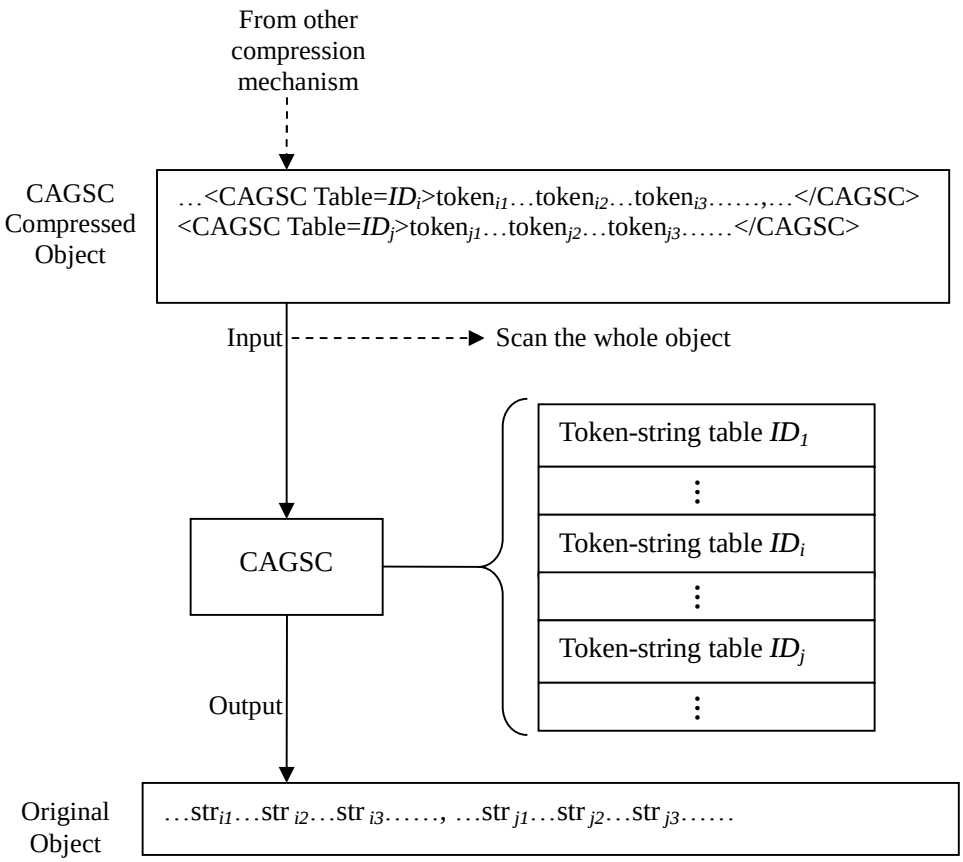


Figure 7.26 Decompression process of CAGSC Compression

From the above process, we see that the starting tag "<CAGSC Table=ID_i>" will guide the decompression program to select the correct token-string table. Even if the CAGSC tokens in different regions are the same, the program will still be able to decompress the tokens to strings correctly. Therefore, we can reuse tokens among different tables and regions without any problem.

7.4.3 Case Study: CAGSC Compression on HTML and JavaScript Strings

In this section, we would like to use HTML tags and JavaScript strings to study the performance of CAGSC compression in web content delivery.

Ideally, CAGSC compression can be applied on any type of special string sets appeared in web content, such as HTML tags, JavaScript strings, css keywords and HTTP headers etc. Here we work on only HTML tags and JavaScript strings because they are the most frequently used special string sets in web content, and they often occupy considerable percentages of object body.

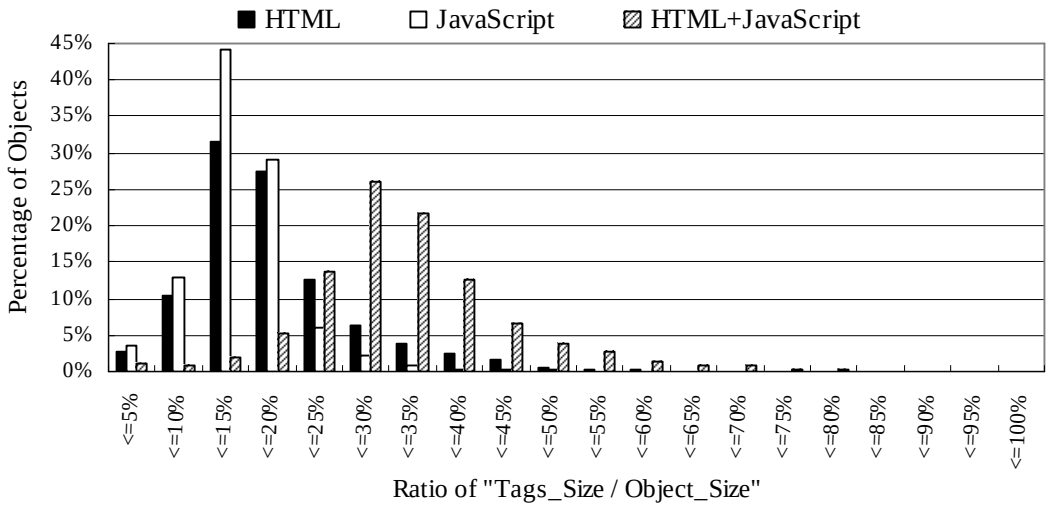


Figure 7.27 Distribution of objects w.r.t. the ratio of "tags size/whole object size"

Figure 7.27 plots the distribution of objects with respect to the ratio of the size of HTML tags and JavaScript strings vs. the size of whole object. From the graph, we see that for the majority of objects, the HTML tags and JavaScript strings all occupy about 5~35% of the whole object size. On average, HTML tags count for about 17.65% of total object size while JavaScript strings occupy about 14.59% of object size. Together, they take up about 32.24% of whole object size. These percentages are quite significant. It gives good potential on performing CAGSC compression on HTML tags and JavaScript strings.

From this graph, we also observe that the curve of JavaScript strings is a little to

the left of the curve of HTML tags. This shows that JavaScript strings occupy relatively smaller portion of objects' body than HTML tags do. This indicates that CAGSC compression may be more effective on HTML tags than JavaScript strings.

As we stated earlier, CAGSC compression should be put to work together with another general-purpose compression algorithm to achieve better overall performance. In our study, we choose to use `deflate` as the general-purpose compression algorithm to work with CAGSC compression. `deflate` is an LZ77 variant and it is a widely used efficient lossless general-purpose compression algorithm. There are a number of programs using `deflate` as the core algorithm, such as gzip and zlib [214] etc. The actual general-purpose compression program we use in this study is zlib. This is mainly because zlib is a free open source package which can be incorporated into our system easily.

When CAGSC compression is working together with zlib, we will use CAGSC compression to first compress an object, and then we pass the CAGSC-compressed object to zlib to do further compression.

7.4.3.1 Selecting Strings for CAGSC Compression

There are about 210 HTML tags and more than 410 JavaScript strings defined in web specifications [35, 289].

In our study, we collect and analyze HTML tags and JavaScript strings by doing analysis on about 34,000 web pages. In total, we get 195 strings in the HTML set and 410 strings in the JavaScript set. To decide the number of strings, what strings and the coding length to be used for each of these two string sets, we compute the weighted frequencies for the strings and the potential gains for certain selections (i.e. subsets) of strings using formulas F7.3 and F7.4.

Table 7.2 and Table 7.3 furnish the potential gains of different selections of

HTML tags and JavaScript strings respectively. From them, we see that the selections of top 128 strings under 1-byte coding give the highest potential gains among all the choices. It is surprising to observe that the selections under 2-byte coding give smaller potential gains although 2-byte coding provides much larger coding space which enables more strings to be selected. Further study reveals the following reasons:

Table 7.2 Potential gains of different selections of HTML tags

	Coding Length			
	1-byte	**2-byte**		
Coding Space	128	32,768	32,768	32,768
Number of strings selected (Remarks)	128 (Only 128 strings are selected due to the limitation of coding space. These 128 strings are those with top weighted frequencies which are greater than zero.)	195 (The full string set of HTML tags. Note that the weighted frequencies of some strings may be less than or equal to zero.)	168 (All the strings whose weighted frequencies are greater than zero are selected.)	128 (Only the top 128 strings from the 168 strings in the left column are selected.)
PG of the selection of strings	99,149,960	62,764,840	68,808,825	68,783,845

Table 7.3 Potential gains of different selections of JavaScript strings

	Coding Length			
	1-byte	**2-byte**		
Coding Space	128	32,768	32,768	32,768
Number of strings selected (Remarks)	128 (Only 128 strings are selected due to the limitation of coding space. These 128 strings are those with top weighted frequencies which are greater than zero.)	410 (The full string set of JavaScript strings. Note that the weighted frequencies of some strings may be less than or equal to zero.)	404 (All the strings whose weighted frequencies are greater than zero are selected.)	128 (Only the top 128 strings from the 404 strings in the left column are selected.)
PG of the selection of strings	111,544,694	86,477,018	86,553,835	85,847,930

(1) Under 2-byte coding scheme, the weighted frequency of a string will be smaller than that under 1-byte coding scheme because a larger coding length is used in the calculation of weighted frequency (see F7.3).

(2) The occurrences of HTML tags and JavaScript strings concentrate on a small subset of the whole string sets. Figure 7.28 plots the CDF of occurrence frequencies of strings with respect to the sizes of string subsets. We see that small subsets

which contain the top 32 or 64 strings already cover the majority of the global occurrence frequencies of all strings. When the subset contains 128 strings, it covers 99.84% of all strings' occurrences in HTML tags and 99.19% in JavaScript strings. This indicates that the 1-byte-coding's 128 coding space provides very good coverage of strings' occurrences of HTML tags and JavaScript strings, which is very comparable to that of 2-byte-coding. So, although 2-byte coding is able to enclose more strings than 1-byte coding, its coverage of occurrence frequencies of strings is extremely similar that of 1-byte coding.

Since 1-byte coding provides good coverage of string occurrences in HTML tags and JavaScript strings and the weighted frequencies of strings are bigger under 1-byte coding, it is understandable why the potential gains on HTML tags and JavaScript strings under 1-byte coding are bigger than that under 2-byte coding.

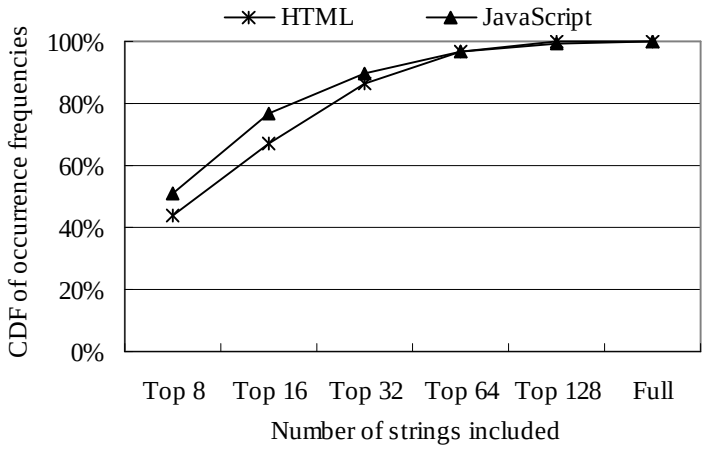


Figure 7.28 Cumulative distribution of strings w.r.t. subset sizes

Based on the above studies of the potential gains, we decide to adopt the 1-byte coding scheme and select the top 128 strings for this case study of CAGSC compression on HTML tags and JavaScript strings in web content delivery.

Table 7.4 lists the top 30 strings among the 128 strings selected from each string set of HTML tags and JavaScript strings.

To look at the average lengths of the selected strings could also help us to

estimate the potential of CAGSC compression.

Table 7.4 Top 30 strings of the selected 128 strings under 1-byte coding

Top 30 HTML tags	Top 30 JavaScript strings
ADDRESS	JAVASCRIPT
WIDTH	DOMAIN
HEIGHT	LOCATION
FONT	CONFIRM
HREF	WIDTH
HTML	WINDOW
BORDER	HEIGHT
TD	BORDER
CLASS	IMAGES
CENTER	OPEN
ALIGN	FOR
FOR	CLASS
CELLSPACING	THIS
CELLPADDING	BGCOLOR
TABLE	IF
BGCOLOR	INDEX
VALIGN	OPTION
IMG	TARGET
TR	VALUE
OPTION	DOCUMENT
SRC	SCRIPT
SIZE	ONMOUSEOVER
COLOR	NAME
FACE	TOP
TARGET	ONMOUSEOUT
STYLE	SEARCH
BR	LANGUAGE
VALUE	TITLE
DIV	RETURN
SPAN	LINK

Taking the frequencies of the strings into consideration, we get the average string lengths and average gain per string for the two string sets. The results are shown in Table 7.5. We see that the average gain per string for both HTML tags and JavaScript strings is substantial.

Table 7.5 Average string lengths and gains under 1-byte coding

	HTML tags	JavaScript strings
Average length of the 128 selected original strings	4.3 bytes	5.3 bytes
Length after CAGSC compression	1 byte	1 byte
Average gain per string	76.74%	81.13%

Also from Table 7.5, we notice that the average length of JavaScript strings is bigger than that of HTML tags. Therefore, the average gain per string for JavaScript

strings is higher than that for HTML tags. However, according to Figure 7.27, JavaScript strings occupies smaller portion of object data than HTML tags do. So, it remains unknown that CAGSC compression is more effective on which string set: HTML tags or JavaScript strings. We will answer this question in the following experimental study.

7.4.3.2 Generating Token-String Tables

After strings have been selected, token-string tables will be generated for them. That is, each string in the selection will be given a token. All the tokens for the strings of one selection (i.e. subset) form a token-string table. Each string table is given a unique ID to distinguish the tokens among tables. The format of token-string tables is as described in subsection 7.4.2.2.4.

In this case study of CAGSC compression on HTML tags and JavaScript strings, we have two string sets. From each set, we select 128 strings. So we have two token-string tables. For simplicity reason, we represent table ID's using integer numbers. For the token-string table for HTML tags, we assign "1" to be its ID; and for JavaScript strings, we assign "2" as the table's ID. We use 1-byte coding scheme in this study, as we discussed earlier. Table 7.6 gives the excerpts of the two token-string tables used in this study.

7.4.3.3 Performance Study

After the token-string tables have been generated, they will be pre-distributed to parties with CAGSC packages. Then we will be able to use CAGSC compression in web content delivery.

Below, we report our experimental study on the performance of CAGSC compression using the token-string tables described in the previous subsection.

To study the performance of CAGSC compression in web content delivery, we

first look at compression ratios that various CAGSC compression mechanisms can achieve. Here we include four mechanisms in our study. They are shown in Table 7.7.

Table 7.6 Excerpts of token-string tables for selected-strings subsets

For selected HTML tags		**For selected JavaScript strings**	
1 (Table ID)	1 (Coding length)	2 (Table ID)	1 (Coding length)
(Token)	(String)	(Token)	(String)
10000001	ADDRESS	10000001	JAVASCRIPT
10000010	WIDTH	10000010	DOMAIN
10000011	HEIGHT	10000011	LOCATION
10000100	FONT	10000100	CONFIRM
10000101	HREF	10000101	WIDTH
10000110	HTML	10000110	WINDOW
10000111	BORDER	10000111	HEIGHT
10001000	TD	10001000	BORDER
10001001	CLASS	10001001	IMAGES
10001010	CENTER	10001010	OPEN
10001011	ALIGN	10001011	FOR
10001100	FOR	10001100	CLASS
10001101	CELLSPACING	10001101	THIS
10001110	CELLPADDING	10001110	BGCOLOR
10001111	TABLE	10001111	IF
10010000	BGCOLOR	10010000	INDEX
10010001	VALIGN	10010001	OPTION
10010010	IMG	10010010	TARGET
10010011	TR	10010011	VALUE
10010100	OPTION	10010100	DOCUMENT
10010101	SRC	10010101	SCRIPT
10010110	SIZE	10010110	ONMOUSEOVER
10010111	COLOR	10010111	NAME
10011000	FACE	10011000	TOP
10011001	TARGET	10011001	ONMOUSEOUT
10011010	STYLE	10011010	SEARCH
10011011	BR	10011011	LANGUAGE
10011100	VALUE	10011100	TITLE
10011101	DIV	10011101	RETURN
10011110	SPAN	10011110	LINK
……	……	……	……
(total 128 entries)	(total 128 entries)	(total 128 entries)	(total 128 entries)

Table 7.7 Four mechanisms for studying compression ratio of CAGSC compression

Abbreviation	**Remarks**
Normal	No compression
CAGSCh	CAGSC compression on HTML tags only
CAGSCj	CAGSC compression on JavaScript strings only
CAGSChj	CAGSC compression on both HTML tags and JavaScript strings

Figure 7.29 plots the improvement that various CAGSC compression mechanisms achieve on size reduction. We see that CAGSC compression can effectively reduce object size by considerable percentage by just compressing HTML tags and JavaScript

strings. The average compression ratios of CAGSCh, CAGSCj and CAGSChj are 85.14%, 86.96% and 78.86%, respectively.

The results in this graph show that CAGSCh have better compression ratio than CAGSCj. In other words, CAGSC compression is more effective on HTML tags than JavaScript strings. The outperforming percentage is 1.2~7.9%. The reason for this phenomenon could be due to the fact that HTML tags occupy bigger portion of object body than JavaScript strings do (refer to Figure 7.27).

When CAGSC compression works on both HTML tags and JavaScript strings (i.e. the CAGSChj mechanism), it produces much better compression ratio than the ones working on one string set only. However, we notice that the size reduction that CAGSChj achieves does not equal to the addition of the size reduction from CAGSCh and CAGSCj. The reason for this is because that there is overlap between the two string sets. When these two strings sets are put together to work for CAGSC compression, the strings that they have in common will be compressed only once. Therefore, the overall improvement is smaller than the total sum of the improvements achieved by working on each single string sets.

Another interesting observation is that the effectiveness of CAGSC compression with different string sets varies for different object sizes. We see that the effectiveness of CAGSCh decreases as object size increases, while this trend is much less obvious with CAGSCj. The reason for this phenomenon may be because that the proportion of different string sets in object body changes in different-sized objects. As object size increases, HTML tags become to occupy relatively smaller portion of object body; so CAGSCh becomes less effective. As for JavaScript strings, its proportion in objects may be quite constant across different-sized objects. Consequently, the effectiveness of CAGSCj is relatively stable. But for very large objects, i.e. those bigger than 128

KBytes, we also see that CAGSCj drops a little in effectiveness. This indicates that the proportion of JavaScript strings in object body also becomes smaller for very large objects. Because the effectiveness of CAGSC compression on HTML tags decreases, so the effectiveness of the compounded CAGSChj compression also decreases, as we can see in Figure 7.29.

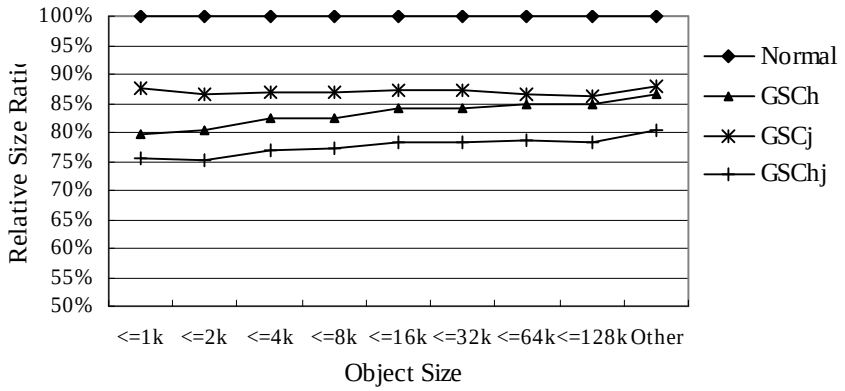


Figure 7.29 Compression ratio of CAGSC compression

Figure 7.29 shows that CAGSC compression can reduce object size by up to 25%. However, there exist some general-purpose compression algorithms such as zlib which work quite well on web content also. Refer back to Figure 7.7, we see that the compression ratio that those algorithms achieve is even higher than CAGSC compression. With this in presence, is it still necessary or beneficial to adopt CAGSC compression in web content delivery? Our answer to this question is positive. This is because: our CAGSC compression is not exclusive to existing general-purpose compression algorithms. In fact, CAGSC compression can be used together with them to further improve the performance of compression in web content delivery. Here we would like to study the effectiveness of CAGSC compression in the presence of another compression algorithm.

As we stated earlier, we use zlib as the general-purpose compression algorithm in our study (please refer to Section 7.4.3 for the reasons). To study the effectiveness of CAGSC compression in working with zlib, we first compress objects using CAGSC

compression, and then apply zlib on the resulting data. In all, we have four compression mechanisms to compare with. They are listed in Table 7.8.

Table 7.8 Four mechanisms for comparison of zlib and CAGSC compression

Abbreviation	Remarks
zlib	zlib compression
CAGSCh+zlib	zlib compression after CAGSCh
CAGSCj+zlib	zlib compression after CAGSCh
CAGSChj+zlib	zlib compression after CAGSChj

Figure 7.30 shows the relative performance of the four compression mechanisms. From the graph, we see that CAGSC compression still achieves considerable improvement when zlib is in presence. Compared against zlib, the relative size ratios of CAGSCh+zlib, CAGSCj+zlib and CAGSChj+zlib are 89.35%, 90.49% and 86.44%, respectively. This can be considered substantial because it is generally agreed that to further improve compression ratio for algorithms like zlib (i.e. `deflate`) is very difficult.

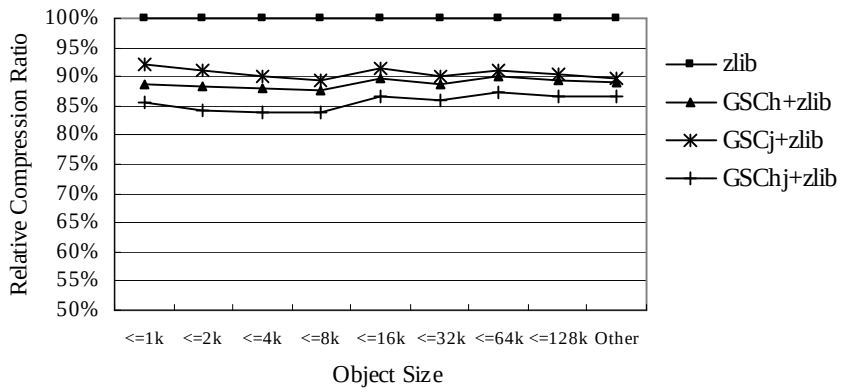


Figure 7.30 Compression ratio of zlib and CAGSC with zlib

An interesting observation is that the CAGSChj+zlib (or CAGSCh+zlib, or CAGSCj+zlib) compression yields the best performance for objects with sizes between 2 KBytes and 8 KBytes. This could be because: (1) For small objects (around 1 KByte), zlib is not very effective in compressing them, so the performance of CAGSChj+zlib is not the best; (2) For big objects (bigger than 8 KBytes), the contribution to size reduction from zlib becomes rather stable, since the contribution from CAGSChj is

getting smaller and smaller as object size increases (refer to Figure 7.29), so the overall performance of CAGSChj+zlib worsens a little.

As we know from the previous sections of Chapter 7, compression can improve object latency and page latency. Below we study the improvement that CAGSC compression can bring in on object latency and page latency. The results presented here are obtained through simulations. The details of experimental environment and test sets are as described in Chapter 5 and Section 7.3.1.

Figure 7.31 and Figure 7.32 plot the improvement of CAGSC compression on object latency against the normal no-compression situation and zlib situation, respectively. In Figure 7.31, we see that the CAGSC compression mechanisms can produce about 4~20% improvement on object retrieval latency. Compared with the improvements that the existing zlib compression has achieved (see Figure 7.3), this improvement is not so significant. However, as we pointed out earlier, CAGSC compression can be used together with zlib as a complementary mechanism to further improve the performance of compression. Figure 7.32 plots the performance of such "CAGSC+zlib" compression mechanisms again zlib. We see that when working with zlib, CAGSC compression does further improve the overall performance on object latency by 4~12%. This is quite substantial since it is generally agreed that to further improve the performance of compression algorithms like zlib is very difficult.

Figure 7.33 plots the improvement of CAGSC compression on page latency against the normal no-compression situation. From this graph, we see that the CAGSC compression mechanisms achieve about 1.5~14.6% improvement on whole page retrieval latency.

We see that the improvement on page latency that CAGSC compression achieves is smaller than that on object latency. In Section 7.3, we have studied this phenomenon

and revealed that (1) the fact that compressible objects in a page (mainly COs) only occupy part of the total page size, (2) dependency among CO and EOs of pages, and (3) limited parallelism width for simultaneous object fetching are the main reasons that prevent compression's performance on object retrieval latency from being translated into page retrieval latency directly.

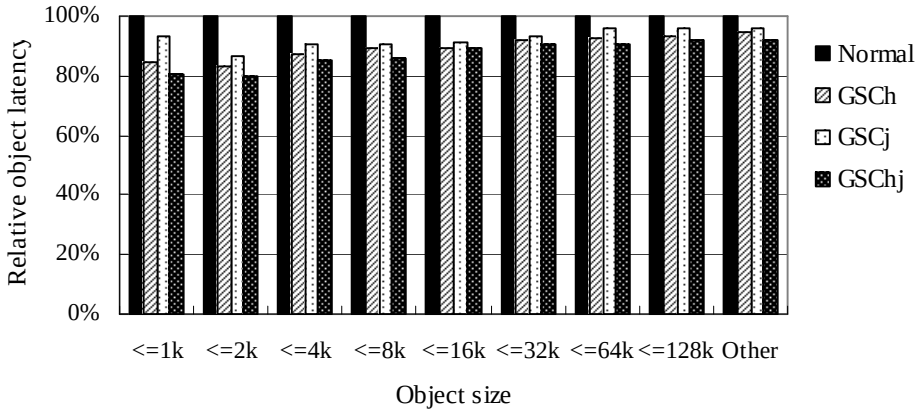


Figure 7.31 Effect of CAGSC compression against normal situation on object latency

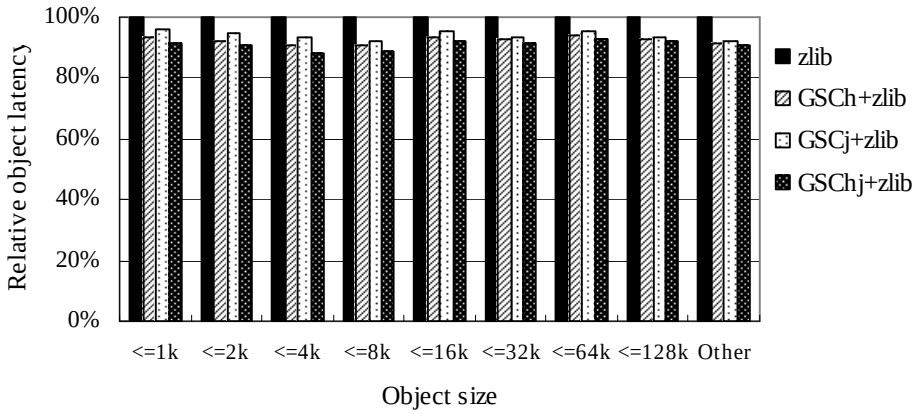


Figure 7.32 Effect of "CAGSC+zlib" against zlib situation on object latency

Because of the above reasons and that CAGSC compression is less effective when working with zlib than being used alone (see Figure 7.31 and Figure 7.32), it is expected that CAGSC compression mechanisms may achieve even smaller improvement on whole page latency when zlib is in presence. Our simulation results show that the "CAGSC+zlib" compression mechanisms can only bring in about 0.8~4.4% improvement on page latency as against zlib compression.

Nevertheless, this is still a non-negligible performance gain, especially when it is obtained on top of an efficient general-purpose compression algorithm since it is rather difficult to further improve the performance of those algorithms.

Furthermore, remember that in this study, our CAGSC compression only works on HTML tags and JavaScript strings. If we include more string sets such as VBScript, css, XML and ESI etc. in CAGSC compression, we would expect that higher performance of CAGSC compression could be seen.

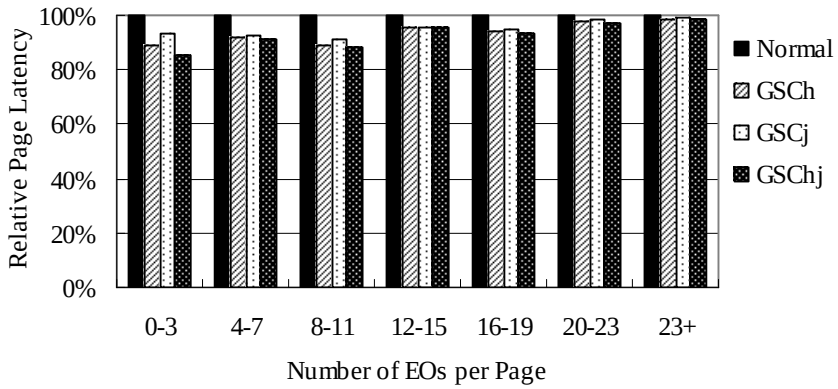


Figure 7.33 Effect of CAGSC compression against normal situation on page latency

7.5 Conclusion

Web compression is an important web acceleration mechanism. This chapter presented our detailed chunk-level study on two major web compression mechanisms, namely pre-compression and real-time compression. The results revealed in our study helps us to have in-depth understanding on the behavior and performance of compression and its effect on page latency. We also propose a novel compression mechanism, named Content-Aware Global Static Compression (CAGSC), to improve the performance of compression in the specific area of web content delivery. Experimental results show that CAGSC can achieve up to 20% and 14.6% improvement on object retrieval latency and page retrieval latency respectively.

Chapter 8 Accelerating Web Page Retrieval through Manipulation of Dependency

8.1 Introduction

Web retrieval latency is always an important issue to web content providers and users. To reduce the latency, two traditional ways are to upgrade the infrastructure of network and servers, and to adopt caching-based acceleration mechanisms such as web caching and prefetching. However, upgrading of infrastructure of network and servers are usually very costly so they are not often used. The caching-based acceleration mechanisms have their limitations and are found not very effective. Therefore, another category of software approaches which aims to accelerate the actual process of web retrieval is getting more and more attention nowadays. Examples in this category include encoding, pipelining and bundling etc. In Chapter 6, we already see that there is good potential of accelerating web retrieval in this direction. In this chapter, we propose innovative acceleration mechanisms which also belong to this category. Our mechanisms try to reduce web retrieval latency by manipulating the dependencies in web retrieval process.

Recall in chapter 4 and 6, web retrieval process consists of a series of objects and operations, and retrieval latency is divided into seven components under WRDM model. While some of the latency components are inherited from operations' execution time and propagation delay of data transmission through network, some others are caused by dependencies between objects and between operations of web retrieval processes. Among the objects or operations in web retrieval processes, there exist some dependencies. For example, the retrieval of EOs relies on the retrieval of the CO of same page. This is because EOs are defined in the body of CO. Dependency would generally introduce latency to retrieval process. The retrieval of an EO can not start until the data chunk containing the definition of the EO has returned to client, this

delay definitely contributes towards the retrieval latency of EOs. As we see, such dependencies are caused by unavailability of information. If such information can be made known to client earlier, the dependency will be shifted to an earlier stage and retrieval latency can be reduced.

In this work, we propose *Information Propagation* mechanisms to manipulate information dependency. These mechanisms could eliminate dependencies or shift dependencies to earlier stages by propagating critical information backward to earlier locations such as previous pages. By doing so, critical information could be made known to client earlier, which would result in improvement in retrieval latency because the dependent operation can now be started much earlier.

The outline of this chapter is as follows. We first discuss the dependency and dependency-introduced latency in web retrieval, and show how the dependency can be manipulated and the relevant latency be reduced. Then we propose two information propagation mechanisms to manipulate two different information dependencies in web retrieval. Following that is the detailed study on the performance of these mechanisms. Then the chapter is concluded.

8.2 Dependency in Web Retrieval and Its Manipulation

8.2.1 Dependency in Web Retrieval

The retrieval processes of web page generally comprise a series of operations. In Chapter 4 and 6, we map such retrieval processes to WRDG graphs under WRDM model. The arcs of WRDG graph represent the relationship between operations. In WRDG graphs, most of those relations are dependency relations since an operation usually depends on the result of its precedent operation in the retrieval process. For example, the network connection arc $a_{c(k,i)}$ denotes the dependency between “server location resolution” operation and “network connection establishment” operation, i.e.,

the establishment of network connection can not be started unless the location of the server has been resolved.

The dependency in web retrieval can be between the retrieval processes of two objects, as well as between two operations in the retrieval process of an individual object. The dependency between objects mainly refers to the relationship between CO and EOs as EOs are defined in CO. This dependency denotes that the requests for EOs cannot be initiated unless the definitions of EOs are made know to client. The dependency between operations appears in the retrieval process of an individual object. In general, every operation would be dependent on the result of its precedent operation in the retrieval process.

Based on the cause of the relationship, we classify the dependencies in web retrieval into two types: (1) Information Dependency, and (2) Happened-before Dependency. If an operation depends purely on some information produced by its previous operation, then the dependency between them is called Information Dependency. Otherwise, the dependency would be treated as Happened-before Dependency. Figure 8.1 gives a classification of the dependencies in WRDG graph.

The dependencies in the retrieval process generally introduce latencies to web retrieval and sometimes such dependency-introduced latency can be significant. We learnt from chapter 6 that web retrieval latency is made up of seven latency components. Those latency components are generally incurred by an operation waiting for the result of its precedent operation, i.e. the dependency between the two operations. For example, the “connection establishment operation” would have to wait for the IP address of the web server being resolved from the “location resolution operation”, and the waiting time is the latency component LRT. From Figure 6.5 and Figure 6.15, we see that some dependencies-introduced latency components are quite

large. For example, the DT time, which is caused by the dependency between CO and EOs, often takes up more than 50% of the object retrieval latency.

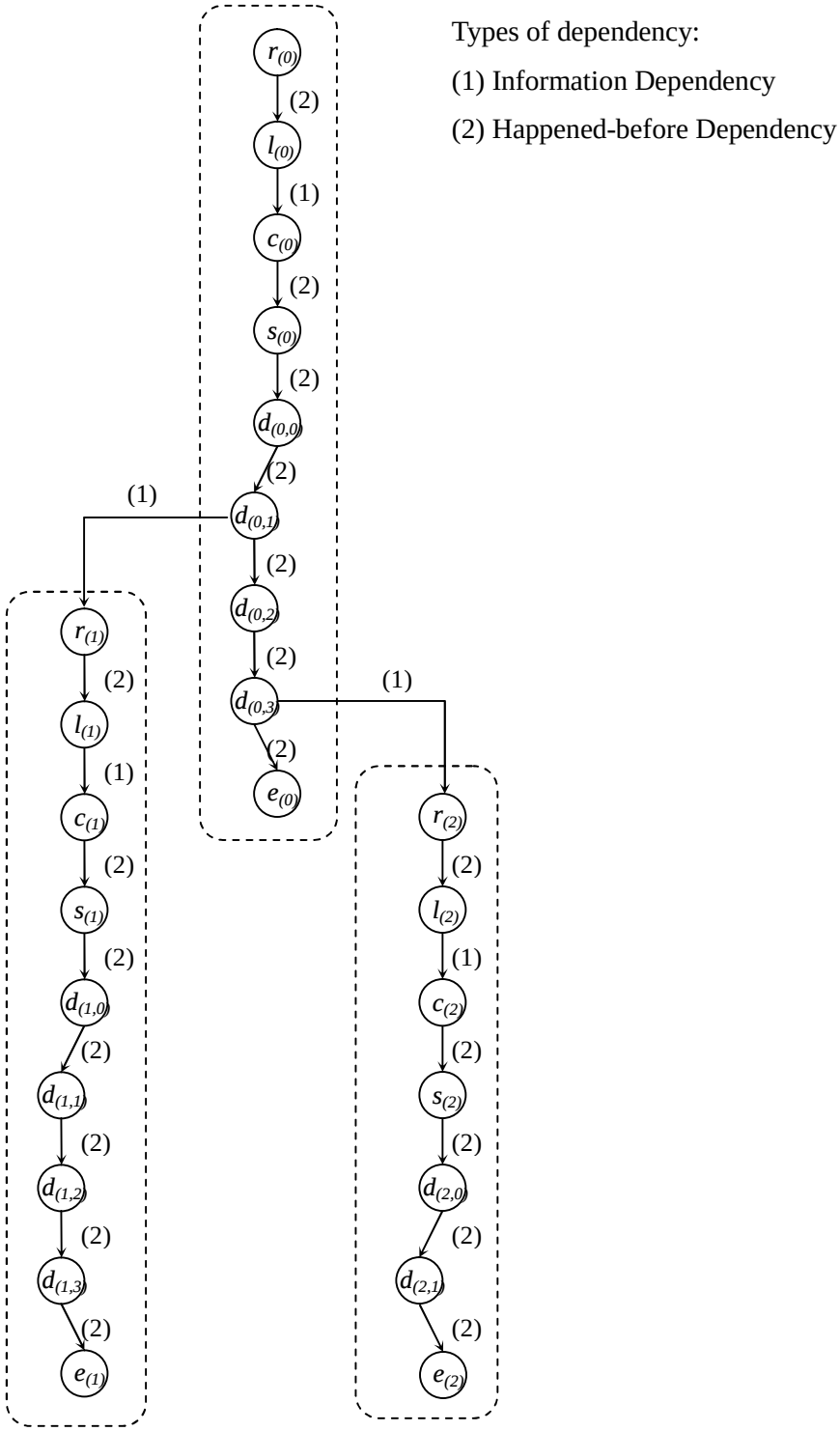


Figure 8.1 Classification of the dependencies in web retrieval

The fact that dependencies have significant influence on web retrieval latency suggests a good direction to improve web retrieval performance by manipulating

dependencies in web retrieval. In this chapter, we try to propose some mechanisms to achieve this goal.

8.2.2 Manipulating Information Dependency in Web Retrieval through Information Propagation

Among the two types of dependencies in web retrieval, the Happened-before Dependency is usually determined by the nature of the operations sequence in web retrieval. For the Information Dependency, it is caused because of the untimely unavailability of information. Because the required information is not made available in time, the starting of the operation which is dependent on that information has to be postponed until the information is ready. If such information can be made available in advance through some mechanism, the dependent operation would be able to start earlier, resulting in faster retrieval process. This prompts us to find a way to make the information required by an operation available earlier, prior to its actual usage.

Since the "entity" that causes Information Dependency is information, which is some type of data, we could try to find some mechanisms to provide such information earlier than it is actually required. The particular mechanism we are going to propose in this chapter is called the *Information Propagation* mechanism. The basic idea of this mechanism is to propagate the critical information which causes dependency to an earlier location/stage in the web retrieval process, and keep it for use when there is need arising. This way, the dependency is shifted to the earlier location/stage. With required information being made ready for use in advance, dependent operations could start executing without any delay, resulting in faster retrieval process.

We would like to also point it out that besides being used to reduce dependency-introduced latency, the propagated information can also be used to enhance the performance of other acceleration mechanisms. For example, pre-connection, persistent connection, bundling, and parallelism schemes etc. all can

make use of the propagated information to further improve their effectiveness.

From Figure 8.1, we see that there are two Information Dependencies in web retrieval based on our WRDM model, i.e. the dependency between the "location resolution operation" and the "connection establishment operation" and the dependency between CO and EOs. The former dependency is between two operations in the retrieval process of an individual object, while the latter one is between the retrieval processes of two objects. We will propose different information propagation mechanisms to manipulate these two dependencies from next section. Results show that our information propagation mechanisms are effective in reducing the latencies incurred by those information dependencies.

8.3 Manipulating the Dependency on Server Location Resolution

In this section, we propose and study a mechanism to reduce the latency introduced by the dependency on location resolution operation. The basic idea is to propagate the location information of servers into web pages and associate it with URLs. Later on, when a client generates a request for an URL, it can use the propagated location information so that the location resolution operation can be eliminated, resulting in faster process for web retrieval.

8.3.1 Dependency on Server Location Resolution

Domain names are widely used in current web systems. The hyperlinks and the definitions of EOs in web pages are generally described in domain-name format. So, the URLs in web requests are also in the format of domain names in most cases. However, to establish network connection with a server, the actual location (i.e. IP address) of the server is required. Thus, the location (IP address) of the server specified in a URL must be resolved prior to the establishment of network connection. The resolution of a server's location is typically a DNS process in current web system, and

it is represented by the location resolution vertex υ_l in WRDG graphs. The operation of establishing network connection highly depends on the result of the location resolution operation, i.e. the IP address of the server. This dependency is a kind of Information Dependency, as we can see it in Figure 8.1.

Location resolution has been the most frequently used process in the Internet, and many technologies such as DNS caching have been developed to improve the performance of this process. Despite the high efficiency of current DNS system, long location resolution time is still often encountered in the web system. Typically, a DNS query which hits in local DNS cache yields very small latency, which appears to be less of a problem for overall web retrieval. However, if a DNS query results in a miss in local DNS cache, the time it takes to get the answer from remote authoritative DNS server will be much bigger, which is no longer negligible to web retrieval. Therefore, it would be preferable if we can have some mechanism to further help in this situation.

The URLs in web requests are originated from different places for COs and EOs. For COs, the URLs are usually originated from hyperlinks in the previous page as web users generally follow the hyperlinks to browse web pages. The URLs in hyperlinks typically have the following format:

```
<A HREF="http://www.nus.edu.sg/"> NUS </A>
```

The URLs for EOs are usually originated from the definitions of EOs within the body of the CO of the same page. There are different types of EOs and the actual format of the definitions for them differs a little. But they generally have similar format as the following one, which is for image EOs:

```
<IMG SRC="http://images.xyz.net/images/1.gif">
```

For the URLs in these two different places, if we can pre-resolve the server locations for them and propagate the information to proper location, then the location resolution

operation in web retrieval may be no long needed, so the location resolution vertex υ_l (and the associated location resolution arc a_l) can be removed from WRDG graphs. As a result, the dependency between the location resolution operation and the network connection establishment operation can be eliminated, which could accelerate the retrieval process as the latency incurred by the location resolution operation has been eliminated.

8.3.2 Server Location Propagation Mechanism (SLP)

In this section, we propose a mechanism to eliminate the dependency incurred by the location resolution operation by propagating the information of server location to eliminate the dependency on the location resolution operation.

The mechanism we proposed to eliminate the dependency on location resolution operation is called Server Location Propagation (SLP) Mechanism. The basic idea of the SLP mechanism is to have web servers to pre-resolve the server addresses for external URLs in the hyperlinks and definitions of EOs in web pages hosted on them. By “external URLs”, we refer to the URLs which specify objects on another server. For external URLs, location resolution operations are usually needed in web retrieval process in order to get the server address for establishing network connection. The pre-resolved server addresses will be propagated into pages and sent to clients so that clients can use them directly for the establishment of network connection without any delay, eliminating the dependency on the location resolution operation.

The SLP mechanism is a server side mechanism. The task of pre-resolving server addresses will be carried out offline by servers during off-peak hours when the servers are relatively idle. After a server pre-resolved the server addresses for the external URLs, it will keep the information in a table named Server Address Table (SAT). The entry of the SAT table typically contains server domain name, server address (IP

address), and the expiry time of the entry. Figure 8.2 shows the structure of the SAT table.

Server Domain Name 1	Address	Expiry Time
Server Domain Name 2	Address1, Address2, …, AddressN	Expiry Time
• • •		

Figure 8.2 Structure of Server Address Table

Note that a server domain name could have multiple corresponding addresses. This is usually for load-balancing on the servers. In this case, the second field of the entry would have multiple addresses.

When the server serves a page, it will look into the page for external URLs and lookup for the corresponding address information in the SAT table. If there is a matching entry, the server will propagate the corresponding address information to the page dynamically. We introduce an optional new tag `ADDR` to HTML for this server address propagation. As described before, the URLs are usually found in two places, i.e. hyperlinks and definitions of EOs. The URLs in these two places before and after the propagation have the formats shown in Figure 8.3.

If the corresponding address for an external URL can not be found in the SAT table, the server will not propagate information for this URL. But the server will record this URL for pre-resolving later on.

In the case where a server domain name has multiple corresponding addresses, the server may base on certain load-balancing algorithm to choose one of the addresses to do the propagation.

At client side, when a user clicks on a hyperlink to go to next page, the browser can directly use the propagated server address associated with the hyperlink (see "Figure 8.3 a)") to establish network connection, removing the need for server location

resolution. When the CO of a page arrives at client, the browser may further trigger the retrieval of the EOs in the page by parsing the CO's body and looking for the definitions of EOs. Since the definitions of EOs (with external URLs) also have associated server address information (see "Figure 8.3 b)"), the browser would also be able to establish network connection immediately without doing the location resolution operation. On the other hand, if the client browser does not understand the ADDR tag, or it choose to ignore this tag, or there is no such ADDR tag at all, it can just proceed as usual with the normal URL to do the retrieval.

a) URL in hyperlink

- Original:

  <A HREF="http://www.nus.edu.sg/"> NUS </A>

- After propagation:

  <A HREF="http://www.nus.edu.sg/" ADDR="137.132.1.10"> NUS </A>

b) URL in definitions of EO (use image EOs as an example)

- Original:

  <IMG SRC="http://images.xyz.net/images/1.gif">

- After propagation:

  <IMG SRC="http://images.xyz.net/images/1.gif" ADDR="137.132.15.10">

Figure 8.3 Propagation of server address

The above process is demonstrated in Figure 8.4.

In the presence of web caching, the above mechanism will not help for those objects which are found in the local cache and still fresh. Because these objects will be served locally, there is no need to contact remote server, so the above mechanism will not be able to help. However, for objects that are not in the cache, or those in the cache but are stale, our SLP mechanism could effectively eliminate the latency component caused by the location resolution operation.

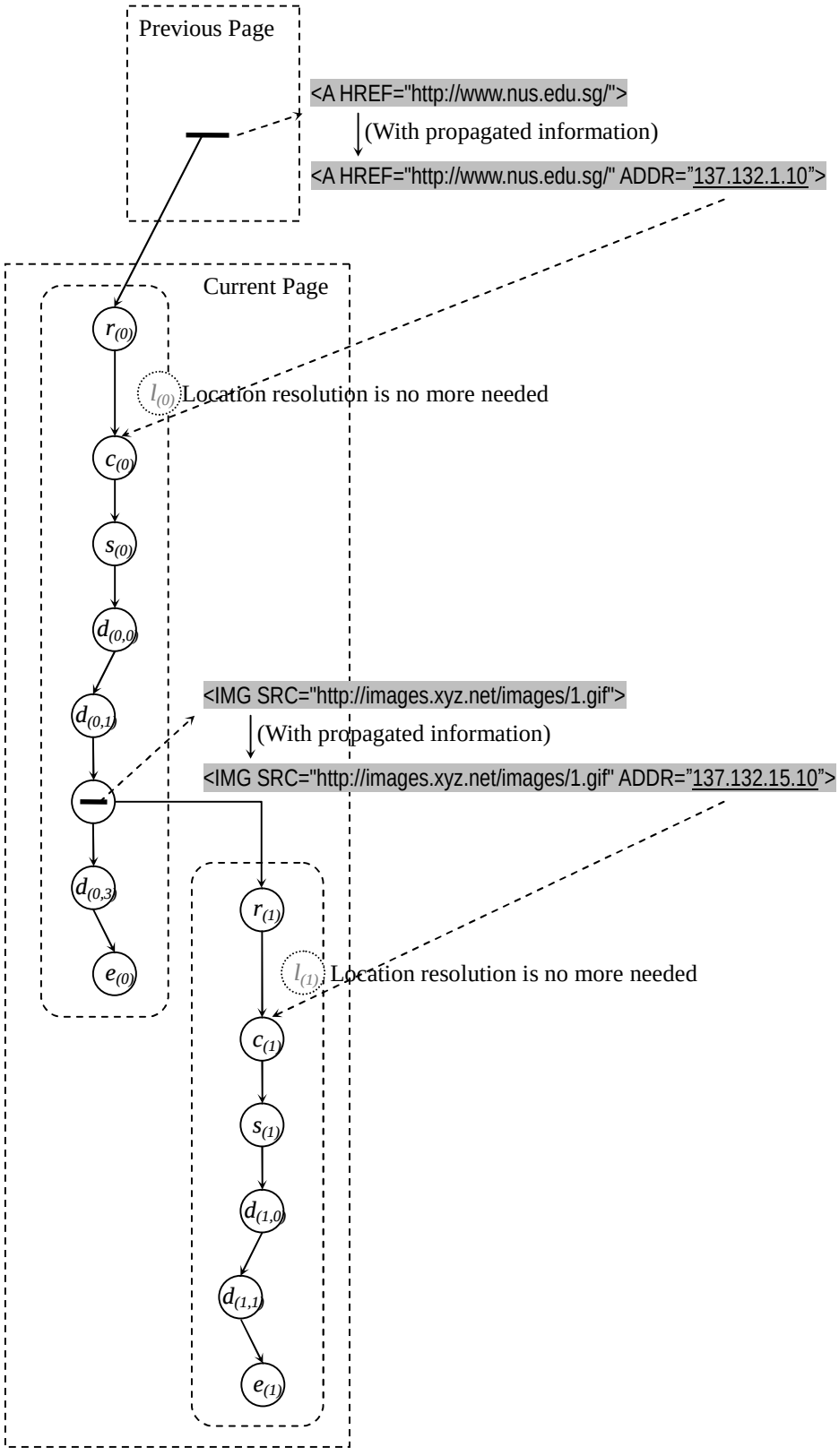


Figure 8.4 Eliminating dependency on server location resolution operation

The SLP mechanism is used to reduce the location resolution latency in web retrieval. It will not affect the correctness of the page content in any means. In the

cases where pages do not have the propagated ADDR tag, or the user's browser does not support this tag, then normal retrieval procedure will be carried out, i.e. resolving the server location, followed by establishment of network connection etc. In any case, the correctness of the web page semantics will not be affected.

8.3.3 Experimental Study

In this section, we report our experimental results on the SLP mechanism. The traces and the simulation systems are described in Chapter 5. In our experimental environment, the local DNS caches exist at the school level (School of Computing) and university level (National University of Singapore). The network traffic in our environment is heavy and diverse. So, the DNS latencies observed in our experiments shall well reflect the experiences of most web users.

First, we would like to investigate the contribution of server location resolution towards object retrieval latency. The server location resolution is typically the DNS process in web system, and the latency for it is the LRT time as discussed in previous chapters. In Chapter 6, we learnt that LRT time usually makes up about 1~6% of object retrieval latency (refer to Figure 6.5). That gives the potential of improvement by reducing this latency component on top of the current DNS system. Although this a few percents contribution is not very significant, it is large enough for being not negligible.

In web system, the result of server location resolution generally falls into the following situations:

1) DNS Local Hit

   These DNS lookups result in hits in local DNS cache, so the server locations can be returned immediately.

2) DNS Local Miss

These DNS lookups result in misses in local DNS cache. They have to contact remote DNS servers to get authoritative answers for the server locations.

3) Numeric

The server location part is already given in numeric IP address form in the URLs. So, there no need to do DNS lookups for such URLs.

4) No IP Available

These DNS lookups cannot find the corresponding server locations in local and remote DNS caches.

5) DNS Negative Hit

These DNS lookups result in hits in local DNS cache, but the results are not server locations, instead, they are messages describing that errors encountered when resolving those server locations in the recent times.

Table 8.1 shows the results of the above five situations. According to the results, about 87% of all DNS lookups resulted in hit in the local DNS caches. When DNS lookups hit in local cache, the DNS latency only contributes less than 1% to the whole object retrieval latency. This confirms the high effectiveness of current DNS system. However, despite of that, we still see about 12% of DNS lookups resulting in misses in the local DNS caches, and the impact of those DNS misses is quite significant: the average latency incurred by them is nearly 2 seconds, which counts for about 20% of the whole object retrieval latency.

There are some URLs in which the server's location part is given in numeric IP address form instead of a domain name. For such URLs, no DNS process is needed. Therefore the LRT time is almost zero. Our SLP mechanism aims to achieve this effect by propagating pre-resolved locations (i.e. IP addresses) in to URLs, so that the requests for those URLs can enjoy minimum LRT time.

There are cases where the DNS processes are unable to resolve the given server domain and return the "No IP Available" result. Such cases usually prolong object retrieval latency very significantly because the DNS system takes long time to contact multiple remote DNS servers to get the final result. In our experiment, this situation usually takes nearly 23 seconds to finish, which is a considerably long time. Luckily, the percentage of requests fall in this category is very small, only about 0.12% in our study.

In our experimental system, the error DNS results such as "No IP Available" will also be cached by the DNS system. If such domain names are submitted for resolving again in the near future, the system could return the result quickly without contacting remote DNS servers. This situation is referred to as "DNS Negative Hit" in our study. From Table 8.1, we can see that the latency for this situation is very small.

Table 8.1 Statistics about server location resolution

	Percentage of requests	**Average DNS latency (second)**	**Average contribution of DNS latency to object retrieval latency**
DNS Local Hit	86.88%	0.002295	0.86%
DNS Local Miss	11.89%	1.964311	20.21%
Numeric	1.00%	0.000084	0.02%
No IP Available	0.12%	22.773561	70.50%
DNS Negative Hit	0.11%	0.007979	67.10%

Some hyperlinks and EOs contained in a page may specify objects on the same server where the current page locates. For the domain names in such hyperlinks and the definitions of the EOs, there is no need to resolve the location since it is the same as current page. For hyperlinks and definitions of the EOs in which the URLs specify different server domain names, location resolution is needed. In our study, we refer to such URLs (in hyperlinks and definitions of EOs) as external URLs. The acceleration of location resolution process is only necessary for external URLs. Therefore, we would like to look into the distribution of such external URLs in web pages.

Figure 8.5 plots the percentage distribution of external URLs in web pages. This

graph reads as follows: Take the category "0~10%" on the X-axis as an example, it shows that for about 44.3% web pages, up to 10% of the hyperlinks in them contain external URLs; and, for about 66.8% web pages, 0~10% of the definition of the EOs contain external URLs. If we put the hyperlinks and the definitions of EOs together, we see that for about 49% web pages, up to 10% of URLs appeared in them are external URLs.

From this graph, we see that there are considerable percentage of URLs appeared in web pages are external URLs. And, the URLs defined in hyperlinks generally have more external URLs than the URLs defined in the definitions of EOs. On average, about 27.9% hyperlinks contain external URLs while about 20.4% definitions of EOs are external URLs. Overall, about 23.9% of all the URLs appeared in a page are external. With this high percentage of external URLs, we would expect that the acceleration mechanisms for location resolution could be effective.

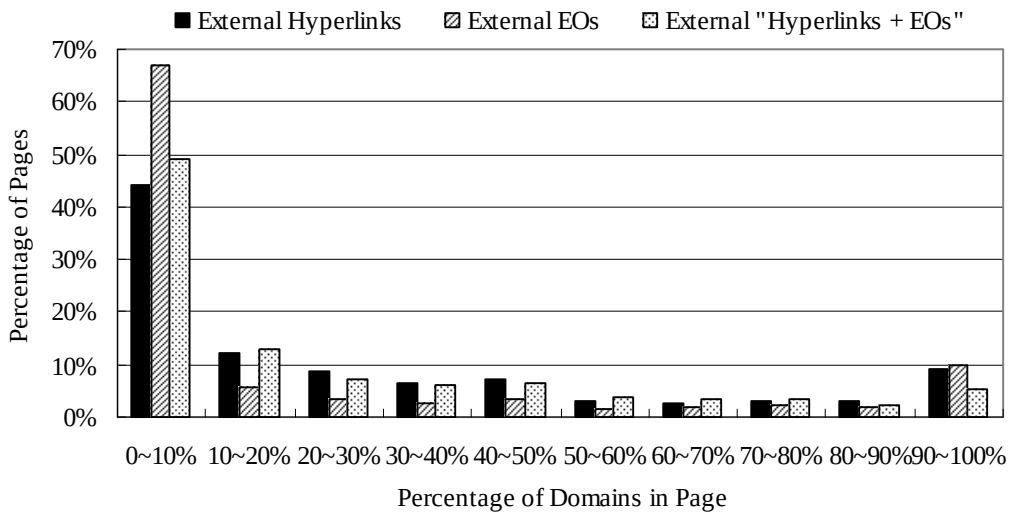


Figure 8.5 Distribution of external domains in web pages

Figure 8.6 shows the distribution of the absolute number of external URLs in web pages. From it, we see that while about 33% of web pages do not contain any external URLs, the majority of web pages do have external URLs defined in them. Moreover, nearly 20% of web pages even contain more than 20 external URLs in them. On

average, there are about 17.6 external URLs in a web page. This big absolute number of external URLs in web pages would also ensure the effectiveness of the acceleration mechanisms for location resolution.

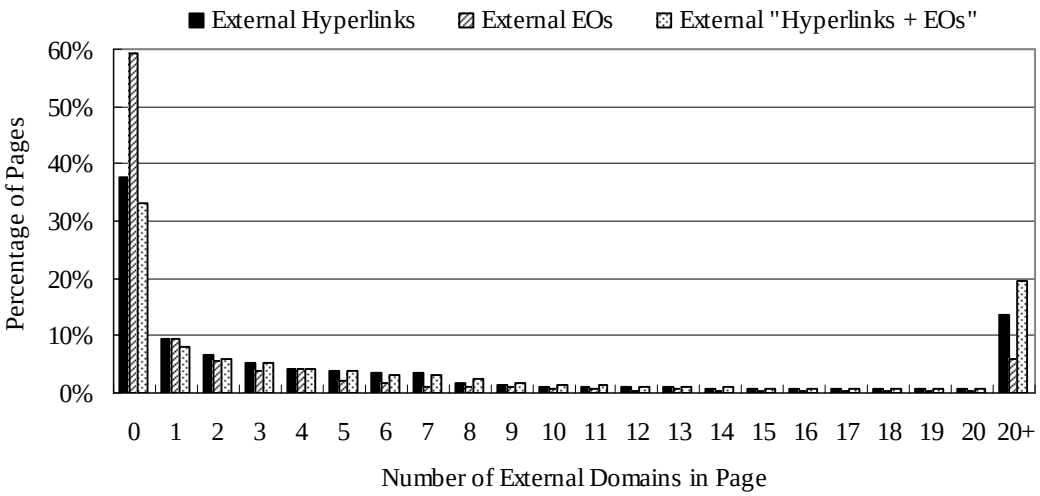


Figure 8.6 Distribution of external domains in web pages

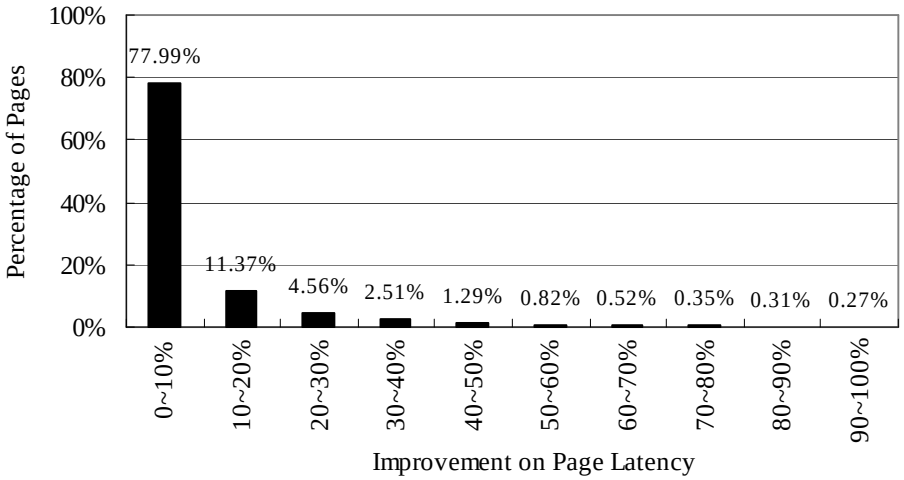
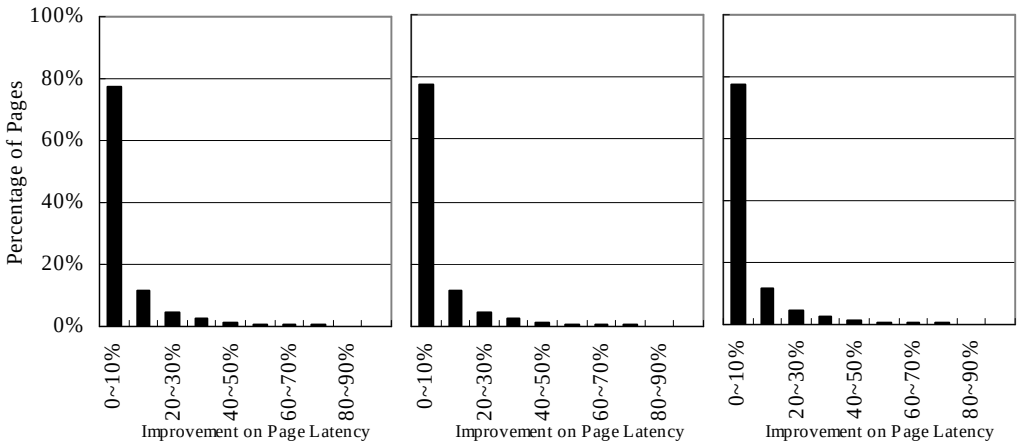


Figure 8.7 Performance of SLP mechanism without caching effect (Parallelism = 4)

Our SLP mechanism tries to reduce the LRT time (i.e. the latency caused by the dependency on server location resolution) by propagating pre-resolved results to URLs in web pages. Figure 8.7 plots the improvement on whole page retrieval latency that SLP mechanism achieves without considering caching effect. From it, we see that SLP can reduce up to 10% whole page latency for about 78% web pages. For other pages, it can achieve even higher improvement. On average, SLP can improve page latency by

about 4.22%. Although this improvement is not very big, it is significant enough for not to be ignored. Furthermore, we note that the improvement is unevenly distributed among pages. If only entry pages are considered, the performance of SLP would be much better because entry pages tend to have more new external URLs and thus they usually spend more time on location resolutions. Our experimental result shows that SLP can achieve about 13.27% improvement on whole page latency for entry pages. This shall be considered as very substantial.

When web caching is taken into consideration, the performance of SLP could be lower since some objects may be found in the local web cache, thus there is no need to resolve server location for them. Figure 8.8 shows the performance of SLP mechanism with considering caching effect. We use three different cache sizes to investigate caching effect on SLP mechanism: (a) 10% of total unique objects size in the trace, (b) 20% of total unique objects size, and (c) Infinite cache size. For the caching replacement algorithm, we use the LRU algorithm, as it is the most popular one used in web caches [73].



(a) Cache Size: 10% (b) Cache Size: 20% (c) Cache Size: Infinite

Figure 8.8 Performance of SLP mechanism with caching effect (Parallelism = 4)

Surprisingly, we see that web caching only has marginal influence on the performance of SLP mechanism. Compared with the average 4.22% improvement

without considering web caching, SLP still achieves average improvements of 4.203%, 4.20% and 4.12% when cache size is 10%, 20% of total unique objects size and infinite, respectively. Further study reveals the following reason for this phenomenon: Web caching will affect the performance of SLP only for those requests which both hit in the web cache and served by SLP. Since the reuse rate of web objects in web cache is low and the percentage of objects served by SLP is also low (recall that only about 12% of object requests result in DNS misses), the overlapping of these two classes of objects will be very small. So, web caching has little influence on the performance of SLP mechanism.

SLP mechanism may bring some overhead to web retrieval. The main overhead concerned in web retrieval performance is the size incensement to the COs of pages. When SLP propagate address to COs, the size of the CO will be increased. Hence, the retrieval latency of the CO may be affected. However, we find that this overhead is negligible to most pages. Because a web page contains about 17.6 external URLs on average, so it takes only about 380 bytes to propagate the address information for them. According to the study in Chapter 6, adding extra 380 bytes to an object has almost no or very marginal impact on the object retrieval latency. Furthermore, the 380 bytes are for 17.6 external URLs. Among these external URLs, there are maybe duplicate ones. SLP only needs to propagate address information for unique external URLs. When we take this into consideration, the size required for propagating address information will be much smaller since the number of unique external URLs should be smaller than 17.6.

From the above study, we can see that the SLP can effectively reduce the latency incurred by the dependency on server location resolution. In general, it can achieve more than 4% improvement on page retrieval latency. Considering this performance

gain is on page retrieval latency instead of object retrieval latency, this improvement shall be considered quite significant. At the same time, SLP retains very marginal overhead to web retrieval performance.

8.4 Manipulating the Dependency between CO and EOs

8.4.1 Dependency between CO and EOs

As we know from previous chapters, a web page is often made up of multiple objects in current web system. Among the objects of a page, one is called the Container Object (CO) and others are called Embedded Objects (EO). The container object of a page is usually an HTML file which contains some content of the page and definitions of the EOs of the page. Embedded objects are usually images, video and audio files etc. The content of both the CO and the EOs must be retrieved and displayed together in order to render the full view of a web page.

There is dependency between CO and EOs in web retrieval process. When a request is created for a page, the URL specified in the request only identifies the CO of the web page. Only when the body of the CO has returned and parsed, will the client know what are the EOs to retrieve. Only after that, will the retrieval processes of EOs be able to be initiated. This indicates that the retrieval of EOs highly depend on the retrieval of the CO. This dependency is captured by the object deriving arcs in WRDG graphs, as we can see in Figure 4.3 and Figure 4.5 etc.

The dependency between CO and EOs introduces an important latency component, Definition Time (DT), to the retrieval latency of EOs. The studies in previous chapters have shown that DT times of EOs often occupy big percentage of object latency, and they contribute significantly towards final page retrieval latency. To reduce this latency component could effectively reduce whole page latency.

The dependency between CO and EOs exists because the EOs are defined in the

body of the CO of the same page. In other words, the cause of the dependency is the definitions of EOs. Such definition is usually a line of HTML code, and typically has the following format (for image type of EOs):

```
<IMG SRC="http://images.xyz.net/images/1.gif">
```

Since the definitions EOs are some kind of information data, so this dependency is also a kind of Information Dependency. From Figure 8.1, we can see that this dependency has been classified as information dependency. Because the cause of the dependency is information, we may again use our information propagation mechanism to manipulate this dependency. From next section, we propose an information propagation mechanism to manipulate this dependency by propagating the definitions of EOs to earlier stage/location of web retrieval process. This way, the dependency is shifted to earlier stage, resulting in significant reduction in the DT times of EOs. Consequently, significant improvement on EOs' latency and whole page latency can be achieved.

8.4.2 Embedded Object Information Propagation Mechanism (EOIP)

In this section, we propose the Embedded Object Information Propagation Mechanism (EOIP) to manipulate the dependency between CO and EOs by propagating the definitions of EOs to earlier stage/location of web retrieval process. We show that our EOIP mechanism could effectively reduce the latency incurred by this dependency, i.e. the DT times of EOs, which in turn significantly improves whole page latency.

The EOIP mechanism is also a server side mechanism. The web server will run a background process during off-peak hour to collect and propagate the information of the EOs for each page. To make it easier to explain the EOIP mechanism, we would like to use an example to describe it. Let us assume we have two pages Page(a) and

Page(b) on a web server, and Page(a) contains a hyperlink pointing to Page(b), i.e. Page(a) contains a line of HTML code similar to the following:

```
<A HREF="http://www.nus.edu.sg/index.html"> NUS </A>
```

When the server runs the background process to do the propagation, it will follow the hyperlink to find the Page(b) (For simplicity reason, we assume this process only looks for pages on the same server. But this process can be extended to include pages on other servers as well). Then the process will parse the CO of Page(b) to find the information of the EOs in it. Let us again assume that Page(b) contains two EOs, i.e. the CO of Page(b) contains two definitions of EOs, like the following:

```
<IMG SRC="http://images.xyz.net/images/1.gif">
<IMG SRC="http://images.xyz.net/images/2.gif">
```

After the server has got the information about the EOs in Page(b), it will propagate such information to Page(a) by appending an optional Embedded Object Declaration Section to the end of the CO of Page(a). Here we introduce an optional new tag `EOD` ("EOD" stands for "Embedded Object Declaration") to HTML for this optional Embedded Object Declaration Section. For the above assumptions, the Embedded Object Declaration Section appended to Page(a) will have the following format and content:

```
<EOD>
http://www.nus.edu.sg/index.html
<IMG SRC="http://images.xyz.net/images/1.gif">
<IMG SRC="http://images.xyz.net/images/2.gif">
</EOD>
```

The first line in the body of EOD section is the URL of a page (here is Page(b)), which comes from the hyperlink in Page(a). The rest lines of EOD section give the definition

of the EOs contained in the page specified by the URL in the first line. The EOD section conveys the information of which EOs are dependent on which CO.

For each distinct hyperlink in Page(a), the server may collect and propagate the EOs information associated with the hyperlink and append a separate EOD section to the CO of Page(a). So, a page may contain multiple EOD sections after the propagation. Each of the EOD section conveys the dependency information between CO and EOs for one distinct page pointed to by a hyperlink. On the other hand, the server will also need to run some processes periodically to check and update the EOD sections to ensure that the information propagated there is valid.

When a client retrieves Page(a), the optional EOD section will be transferred to the client after the original content of Page(a) has been transferred. There are generally idle times between page visits (due to user's viewing a page, or the post-processing of a page such as compiling Java programs). These idle times can be used to transfer the optional EOD sections, so the extra transfer latency would be hidden and thus not perceivable to web users. (In other words, the `EOD` section is like a section of piggyback data which is sent along with the page after the original page content finishes.)

The propagated information will be used by client to reduce the latency incurred by the dependency between CO and EOs. Web users frequently follow hyperlinks to browse web pages. When a user clicks on the hyperlink in Page(a) to go to Page(b), the web browser will get the URL from the hyperlink to create a request for Page(b). At the same time, it would also look up in the EOD sections to see if there is any one containing this URL. If there is a match, the browser will get the definition of the EOs contained in Page(b) from the EOD section and start to retrieve them without waiting for the CO of Page(b) to be returned from the server. In other words, the requests for

the EOs can be triggered as early as the request for the CO of the same page.

The process of EOIP mechanism is illustrated in Figure 8.9. From this graph, we can see clearly that the dependency between the CO and the EOs of a page has been shifted to a much earlier location/stage in web retrieval, i.e. the EOD sections in the previous page, and the latency caused by this dependency, i.e. the DT times of EOs, has been effectively eliminated. As a consequence, whole page retrieval latency can be improved significantly.

If web caching is adopted, the above EOIP mechanism may not help much for those pages whose COs are found in the local cache and still fresh. For such pages, their COs can be server rapidly from the local cache, so it takes little time for the browser to get their content and discover the EOs defined in them. In this situation, our EOIP mechanism will not help much. However, for pages whose COs are not in the cache, or they are in the cache but stale, our EOIP mechanism would be very effective in reducing the latency caused by the dependency between CO and EOs.

Note that the EOIP mechanism is only intended to manipulate the dependency between CO and EOs to reduce the relative latency. The `EOD` section is only an optional section in HTML files. In the case where pages do not have such propagated `EOD` section, or there is no enough time between page visits for transferring this section, or the user's browser does not support this section, then the browser can just ignore it and the normal retrieval procedure will be carried out. In all the cases, the correctness of the web page semantics will not be affected in any way.

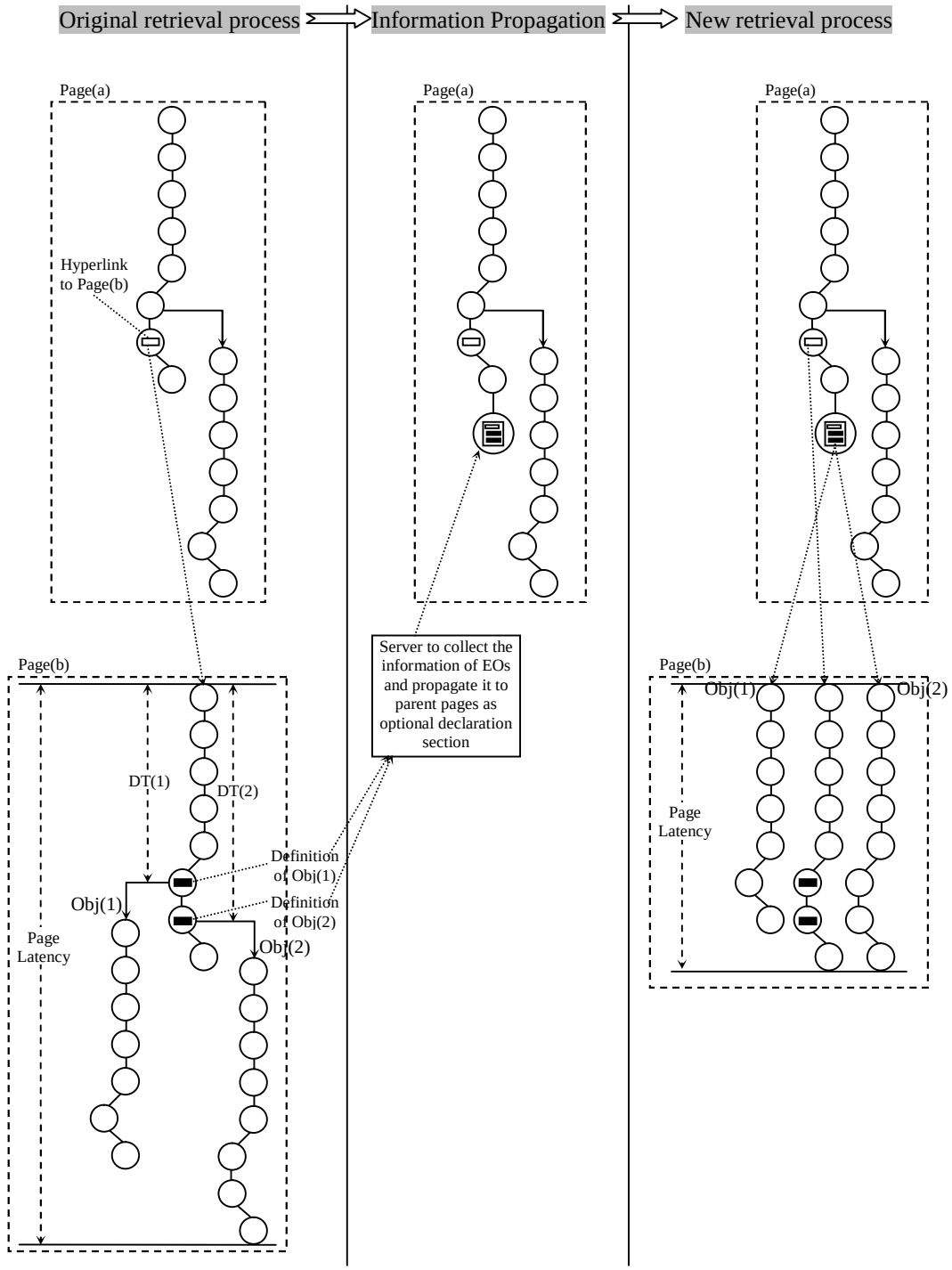


Figure 8.9 Eliminating dependency between CO and EOs

8.4.3 Experimental Study

We have conducted trace-driven simulations to study the performance of our EOIP mechanism. Our experiments rely on very detailed information about web retrieval, such as the number and times of data chunks, definition points of EOs etc. Such information is not available in existing traces. In order to obtain the information, we conducted real retrieval for a large number of web pages and recorded detailed operation and chunk level logs. The tools and environment are described in Chapter 5.

EOIP aims to reduce the latency caused by the dependency between CO and EOs, i.e. the DT times of EOs. In Chapter 6, we already showed that the majority of pages contain multiple EOs (on average, a page contains about 13.5 EOs), and the DT time of EOs often occupies more than 50% of their retrieval latency (refer to Figure 6.12 and Figure 6.15 etc.). This indicates a high potential on latency reduction by reducing DT times of EOs.

Figure 8.10 plots the performance of EOIP mechanism without considering web caching effect. From it, we can see that EOIP can improve page retrieval latency considerably. On average, EOIP can achieve about 10.66% improvement on whole page retrieval latency.

Here we have an interesting observation. At first, we speculated that EOIP may perform better for web pages with more EOs. However, we noticed in Figure 8.10 that the improvement of EOIP does not seem to increase as the number of EOs in a page increases. Further study reveals that this could be due to the following reasons:

1) The absolute improvement of EOIP does increases with the increasing number of EOs in a page. However, for pages with more EOs, their whole page latency often increases, too, and even more significantly. Therefore, the relative improvement of EOIP does not increase.

2) Parallelism width limits the improvement of EOIP. EOIP reduces the DT times of EOs, which means that EOs will be made known for retrieval very early and simultaneously. For pages with many EOs, the default parallelism width of 4 will become insufficient for retrieving all the EOs, so many of the EOs' requests will be held in waiting state. In other words, limited parallelism has bottlenecked the performance of EOIP.

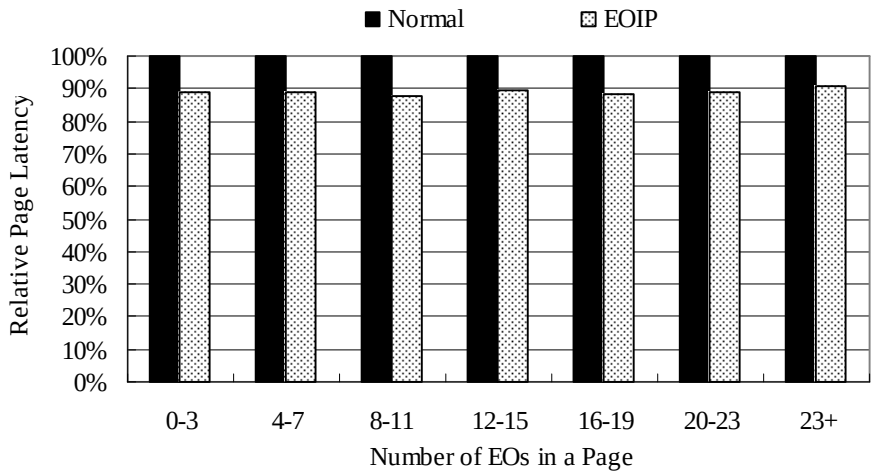
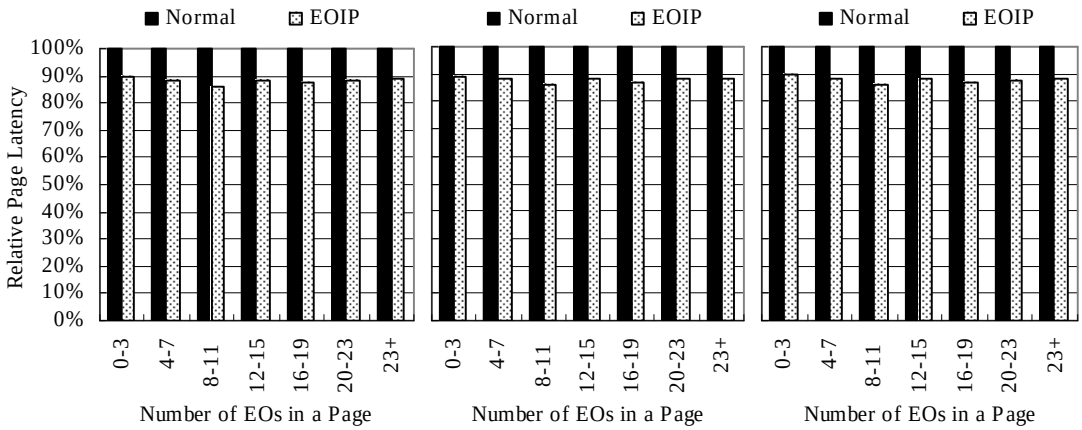


Figure 8.10 Performance of EOIP without caching effect (Parallelism = 4)



(a) Cache Size: 10% (b) Cache Size: 20% (c) Cache Size: Infinite

Figure 8.11 Performance of EOIP with caching effect (Parallelism = 4)

Web caching could have some impacts on the performance of EOIP. Figure 8.11 shows the performance of EOIP mechanism with the presence of web caching. Again, three different cache sizes are used in our study, i.e. (a) 10% of total unique objects size in the trace, (b) 20% of total unique objects size, and (c) Infinite cache size.

From the graphs, we see that EOIP seems to perform even better when web caching is taken into consideration. Compared with the average 10.66% improvement without considering web caching, EOIP achieves average improvements of 11.72%, 11.76% and 11.8% for the three cache sizes. This is surprising as we thought web caching could lower the effectiveness of EOIP. After some further study, we find the following reasons for this phenomenon:

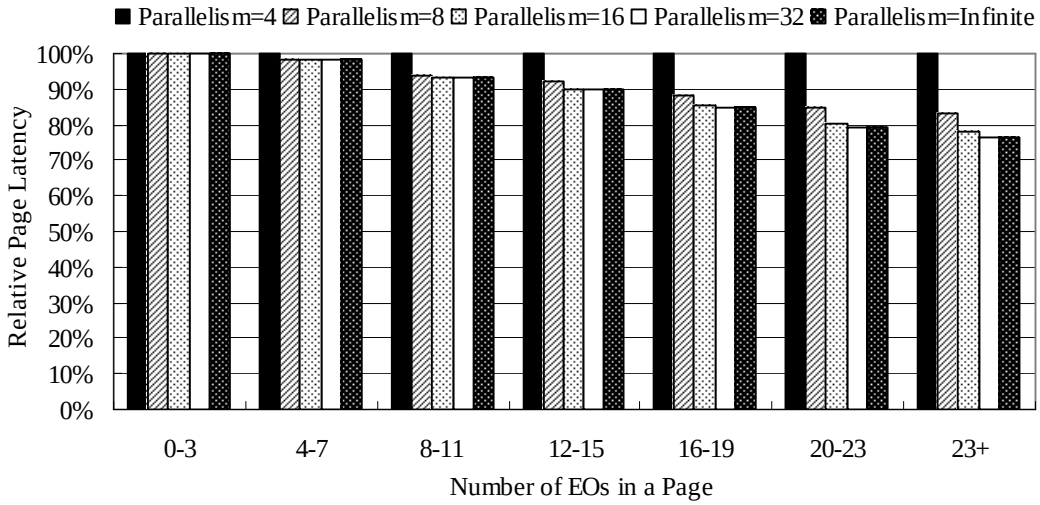
- Cache hit ratio is low. So the impact of web caching on the performance of EOIP is limited.
- Parallel fetching is employed in web retrieval, which makes it possible for the retrieval of an object to virtually have no effect on the whole page latency. So, for some objects that are hit in web cache, they may not have much impact on whole page latency.
- As EOIP makes EOs ready for retrieval early and simultaneously, it puts high demand on parallelism. When web caching is in presence and some EOs are hit in the cache, the number of objects being held in waiting state could be reduced. This could have positive effect on whole page latency. To give an evidence of this reason, let us look at the first data row in Table 8.2. This table gives the detailed data used for plotting Figure 8.11. The result in this row shows the performance of EOIP on pages with 0~3 EOs. For such pages, the default parallelism width of 4 is sufficient. We see that for this situation, the performance of "with caching" is actually worse than that of "without caching". This contrary example shows that our above analysis could be correct.

As we mentioned earlier, parallelism width could limit the performance of EOIP. Next, we would like to investigate the effect of parallelism width on EOIP. Since we are to investigate the effect of different parallelism, we fix the cache size to infinite in this

study. This way, the effect of different cache sizes would be eliminated from the results.

Table 8.2 Performance of EOIP without/with caching effect (Parallelism = 4)

Number of EOs in a page	Performance (without caching)	Performance (with caching)		
		Cache Size: 10%	Cache Size: 20%	Cache Size: Infinite
0-3	***88.97%***	***89.37%***	***89.40%***	***89.59%***
4-7	88.73%	88.01%	88.03%	88.06%
8-11	87.65%	86.01%	86.02%	85.98%
12-15	89.40%	88.54%	88.40%	88.36%
16-19	88.54%	87.24%	87.17%	87.13%
20-23	88.96%	88.28%	88.14%	87.89%
23+	90.62%	88.62%	88.55%	88.37%



(Cache Size = Infinite)

Figure 8.12 Performance of EOIP under different parallelism width

Since EOIP makes EOs ready for retrieval early and simultaneously, it will put high demand on parallelism width. So, a wider parallelism could possibly make EOIP more effective in reducing page retrieval latency.

Figure 8.12 gives the performance of EOIP under different parallelism width. Here we only investigate parallelisms wider than 4 since the parallelism width in most current web browser is 4.

As expected, we see that EOIP performs better under wider parallelism width. Compared with the performance under the commonly used parallelism width of 4,

EOIP's can improve page retrieval latency by 8.87%, 11.24%, 11.78% and 11.85 on average when parallelism width is 8, 16, 32 and infinite respectively. The impact of parallelism width on the performance of EOIP is generally bigger for the pages with more EOs. This is understandable because there will be more objects to make use of the wider parallelism in such cases, so EOIP becomes more effective.

In terms of web retrieval performance, the major overhead that EOIP introduces is transferring the extra `EOD` section to clients. Here we would like to study the impact of such overhead on web retrieval performance.

From Chapter 6, we know that the average chunk size is 5.3 KBytes (see Figure 6.7). Normally, this size could be enough for the propagation of information for about one hundred EOs. If we propagate information of hundreds of EOs into a page, they may only bring in a small number of extra chunks. From Figure 6.8, we know that the time for transferring one chunk is only about 0.01~0.02 seconds. So, the time for transferring the extra chunks should not be significant. Moreover, the propagated information composes an optional section of CO's body and such optional section is supposed to be transferred during the idle times between pages. If the idle time is big enough, then the transfer time for those extra chunks would be hidden by the idle time and thus not perceivable to web users.

Figure 8.13 plots the distribution of idle times between page accesses. From this graph, we see that the majority of intervals between page accesses are between 1 to 64 seconds. On average, the idle time between pages is about 92.6 seconds, which should be considered as more than sufficient for transferring the optional `EOD` section for most web pages. In other words, the latency incurred by transferring the extra chunks (which are used for the propagated information) can easily be hidden by the idle times between page accesses. Therefore, the overhead introduced by EOIP can largely be

ignored in reality.

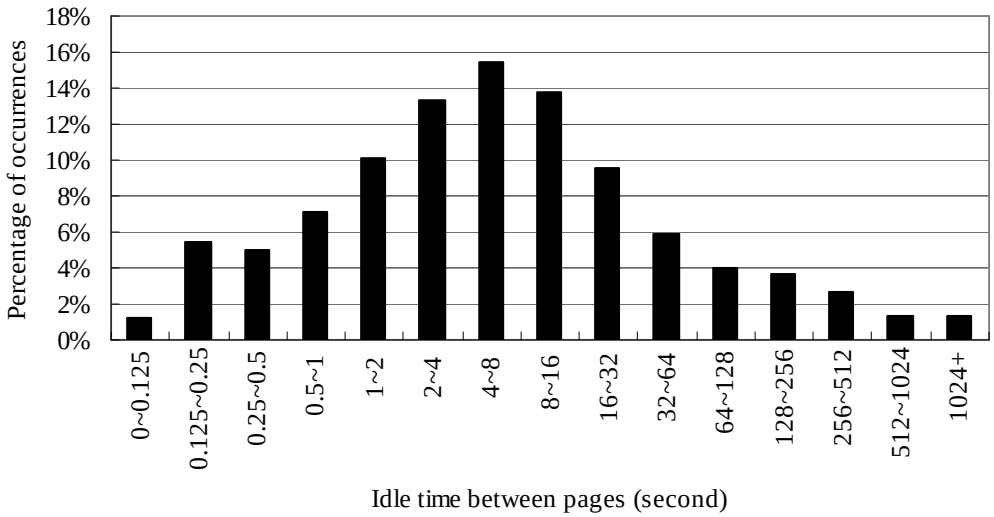


Figure 8.13 Idle times between page accesses

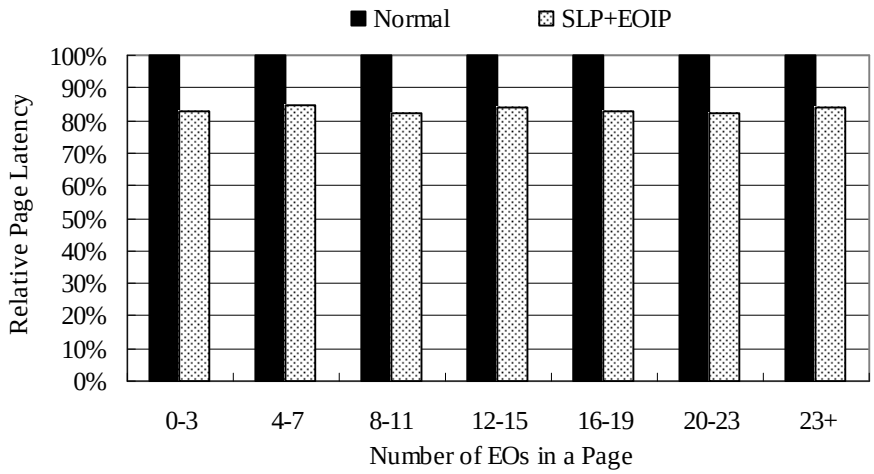


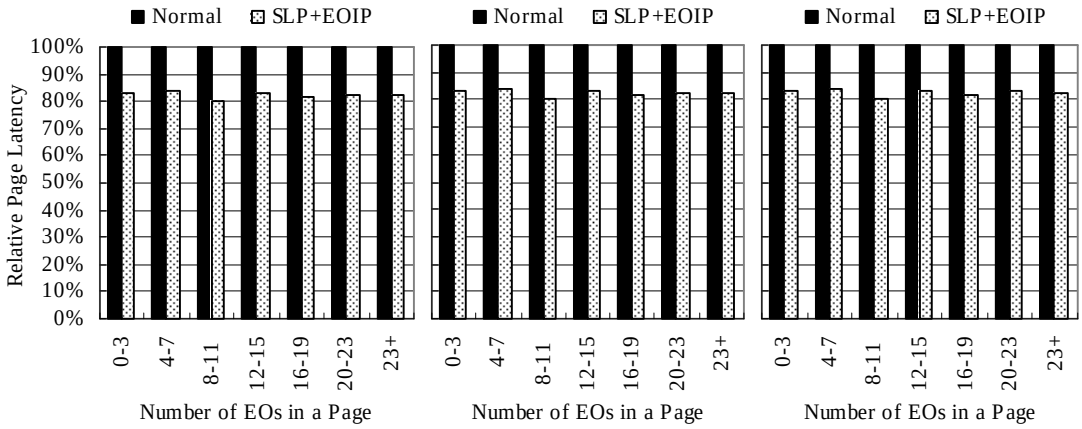
Figure 8.14 Performance of SLP+EOIP without caching effect (Parallelism = 4)

8.5 Effect of Integrated SLP and EOIP Mechanism

In the last two sections, we propose and study two mechanisms for manipulating the dependencies in web retrieval, i.e. the SLP and EOIP mechanism. Our results show that these two mechanisms are effective in improving web retrieval latency. Because SLP and EOIP are independent from each other, so they can be used together. In this section, we would like to complete this study by studying the performance of the integrated SLP+EOIP mechanism.

Figure 8.14 gives the overall performance of SLP+EOIP without considering

caching effect and under the default parallelism of 4. We see that the integrated SLP+EOIP mechanism can achieve about 15.63% improvement on average, which is much better than the individual SLP and EOIP mechanism. This indicates that these two mechanisms actually reinforce each other when they work together.



(a) Cache Size: 10% (b) Cache Size: 20% (c) Cache Size: Infinite

Figure 8.15 Performance of SLP+EOIP with caching effect (Parallelism = 4)

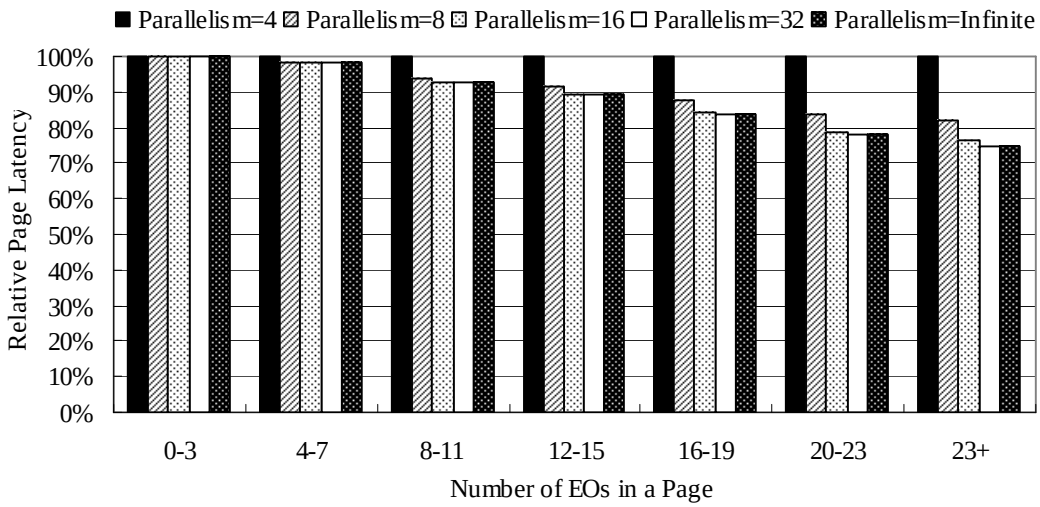


Figure 8.16 Performance of SLP+EOIP under different parallelism width

When web caching is taken into consideration and the parallelism width varies, the integrated SLP+EOIP mechanism still continuously outperforms the individual SLP and EOIP mechanism in all situations. Figure 8.15 plots the performance of SLP+EOIP with considering caching effect, and Figure 8.16 shows the performance of it under different parallelism width. We see that the integrated SLP+EOIP mechanism

can achieve about 16.53%, 16.54% and 16.75% improvement when cache size is 10% and 20% of total unique objects size in the trace and infinite respectively. When parallelism width is increased from 4 to 8, 16, 32 and infinite, the integrated SLP+EOIP mechanism yields performance gain of about 9.49%, 12.07%, 12.64% and 12.8% respectively,

Overall, the performance of the integrated SLP+EOIP mechanisms is better than the individual SLP and EOIP mechanism. This is expected as these two mechanisms address two different dependencies in web retrieval. Their effects are quite independent from each other. So, when they are put together, they would reinforce each other. This leads to even better performance gain than the individual mechanism.

8.6 Conclusion

In this chapter, we analyzed and studied the dependencies in web retrieval and the latencies introduced by them. We showed how such dependency-introduced latencies can be reduced through manipulation of the dependencies. We proposed two innovative information propagation mechanisms, namely SLP and EOIP, to manipulate two different latencies in web retrieval. We conducted simulations to study the performance of SLP and EOIP. Our results show that these two mechanisms can reduce about 4.22% and 10.66% of whole page retrieval latency. The integrated SLP+EOIP mechanism can achieve about 15.63% improvement on page retrieval latency. This shows that information propagation can be an effective method to improve web retrieval latency.

Chapter 9 Exploiting Fine-Grained Parallelisms for Acceleration of Web Retrieval

9.1 Introduction

With the advent of cheaper and faster processing power and storage, there has been a wide-spread proliferation of digital documentation, multimedia materials and web-based applications in the Internet. More and more web pages tends to comprise such digital materials like image files, pdf files, flash animation files, video and audio files, application executables and so on. As these digital files are usually considerably big in size, this trend has unfortunately meant that web surfers are increasingly being overwhelmed by large objects. Large web objects normally take long to transfer and they are often the dominating performance bottleneck of retrieval latency for web pages containing them. With the increasing number of large objects being used in web pages, the need to reduce the retrieval latency of web pages becomes even more imperative.

When a web page contains large web objects, the latency component CST will clearly become the main dominating factor to page latency. Figure 9.1 gives an illustration WRDG graph for the retrieval process of a page with large objects. For such pages, accelerating the retrieval process of the large objects would effectively reduce the whole page latency. To cope with the long CST latency, the increase of network bandwidth and content encoding schemes are often proposed. However, increasing network bandwidth is very costly and has its limitations; content encoding schemes are generally not possible or effective to digital documentation and digital multimedia materials. This situation prompts us to study a new mechanism to solve the problem, which is fine-grained parallelism for parallel fetching of multiple sub-ranges of a large object.

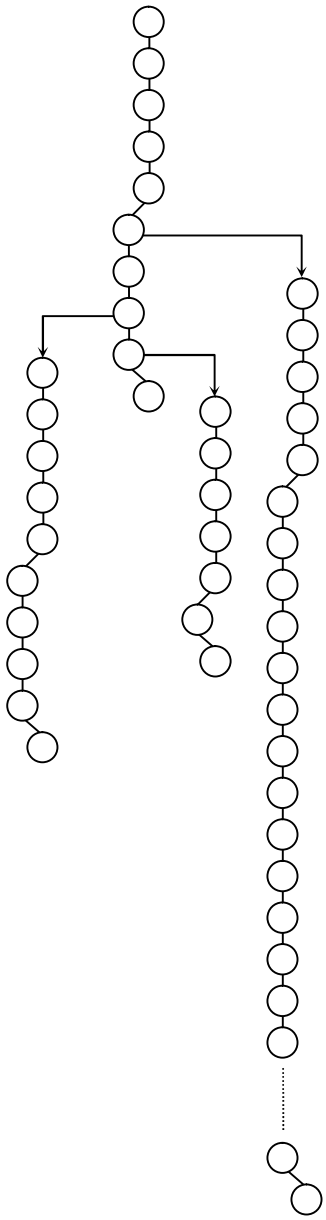


Figure 9.1 Retrieval process of a page with large object

Parallelism is not new in web retrieval. Most common web browsers already employ parallelism for concurrent retrieval of objects. For example, both MS IE and Netscape use parallelism of four to retrieve multiple objects in a page [290]. In Chapter 6, our studies already show that parallel fetching of objects can significantly reduce page latency since most web pages contain multiple objects. But that parallelism is at object-level, i.e. the parallelism only exists among the retrieval of objects. In this chapter, we extend the concept of parallelism to a very fine-grained level in web retrieval, i.e. the intra-object level, to accelerate the retrieval process for large web

objects. Since the intra-object level is mainly characterized by data chunks transfer, we hence call our mechanism the *Chunk-Level Parallelism* (*CLP*) for the acceleration of web retrieval.

A large object is typically transferred in a series of large number of chunks of data. If we can divide this large series of chunks into multiple smaller sub-series and retrieve them in parallel, the retrieval latency of large objects could be reduced considerably, which would finally result in improvement on whole page latency. The idea has been made feasible and practical with the evolution of HTTP protocol. In HTTP/1.1, it introduces the concept and support for “partial object”. It allows a client to request only a partial body of a web object by using the “Range” header. Objects can be broken down into multiple sub-ranges according to various structural units (currently, the only range unit defined by HTTP/1.1 is "bytes") [37].

There is very little related work in this direction in current literature. This situation motivates us to conduct detailed study on this problem. In this chapter, we exploit and perform comprehensive study and analysis on the effect of CLP on web retrieval latency. Our study reveals some complicated issues regarding chunk-level parallelism, which shows that the application of this mechanism is not so easy and straight-forward as people might have thought. We conduct simulation experiments as well as real system testing to study the performance of CLP. Our results show that CLP can achieve significant improvement on object retrieval latency and whole page latency when large objects are in presence.

The rest of this chapter is structured as follows. We first analyze the demand for chunk-level parallelism in web system. Then we describe the mechanism of chunk-level parallelism and study the issues related to it. Next we study the performance of chunk-level parallelism to illustrate its effectiveness. Following that,

some discussion on the system implementation considerations is given. Then the final section concludes this chapter.

9.2 Exploiting Chunk-Level Parallelism

9.2.1 Demand for Chunk-Level Parallelism

Large web objects often have dominating effects on whole page retrieval latency. When a web page contains large web objects, the retrieval latency of those objects is often the dominant factor to the whole page latency. So, we would like to first investigate the presence of large objects in web pages. Figure 9.2 plots the distribution of web pages with respect to the largest object size they contain.

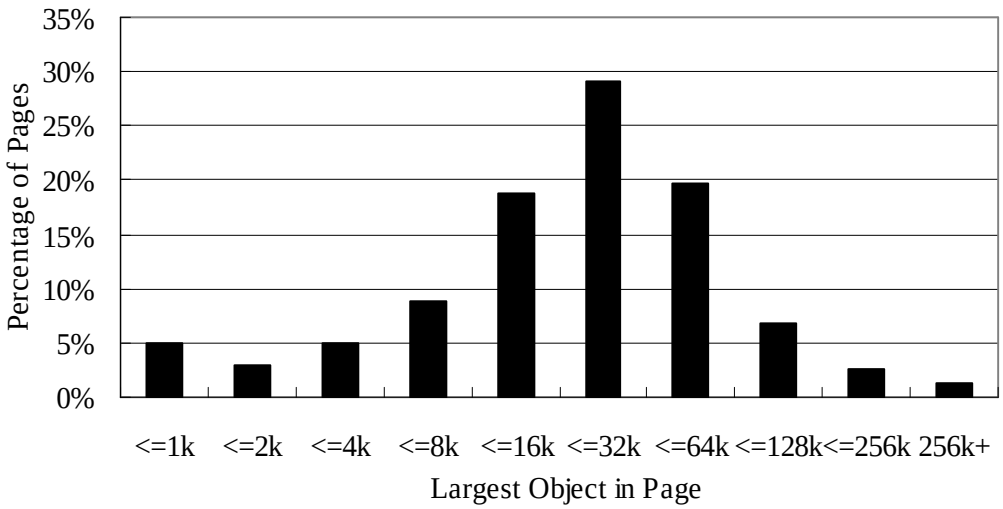


Figure 9.2 Distribution of pages w.r.t. size of the largest object in the page

From this graph, we see that the majority of web pages have objects not bigger than 64 KBytes. However, there are about 10.54% of web pages contain objects bigger than 128 KBytes, which could prolong page retrieval latency significantly. As digital documentation, multimedia materials and web-based applications etc. are increasingly distributed over the web, we expect to see the percentage of web pages containing large objects to continuously increase in the future. Nevertheless, the percentage of 10.54% is already significant enough for us to look into effective mechanisms to

accelerate the retrieval process of them.

Figure 9.3 depicts the distribution of types of large objects, and Table 9.1 gives more detailed information for those "xxxx" types shown in the graph. Besides the traditional object types such as image, text and multimedia files, we also see that there is a type "application", which occupies a large percentage of the distribution. This type includes some subtypes like pdf, shockwave-flash and executables. As the digital documents and web-based applications become more and more popular, we expect to see the amount of objects of this type to grow considerably in the future.

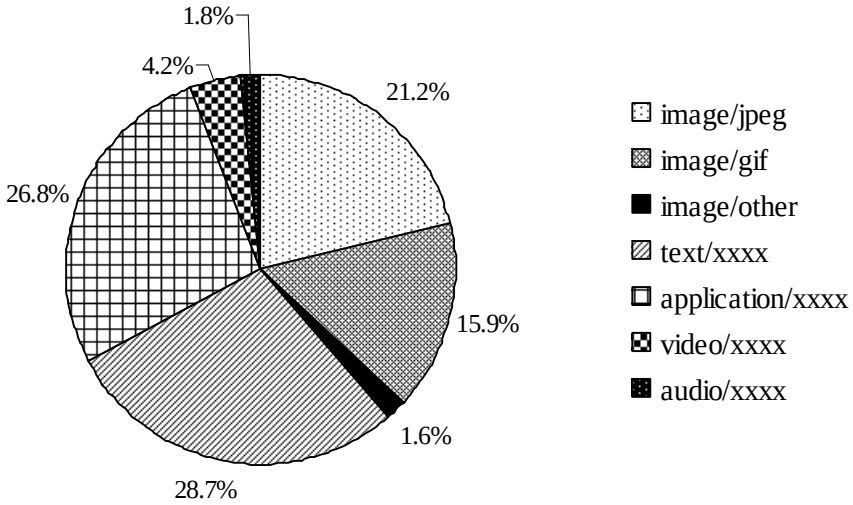


Figure 9.3 Distribution of types of large objects

Table 9.1 Detailed object types

image/other	text/xxxx	application/xxxxxx	video/xxxxxx	audio/xxxxxx
image/bmp image/x-ms-bmp image/png	text/html text/plain	application/octet-stream application/cache-digest application/pdf application/postscript application/vnd.ms-asf application/x-ipix application/x-director application/x-shockwave-flash	video/mpeg video/quicktime video/x-ms-asf video/x-ms-wmv www/unknown	audio/midi audio/mpeg audio/wav audio/x-ms-wma audio/x-pn-realaudio audio/x-realaudio audio/x-wav

As we know, object data are transferred in a sequence of data chunks, and the chunks often have limited size. In Figure 6.7, we learnt that the majority (65%) of chunks have sizes between 1 KBytes and 2 KBytes, and the average chunk size is

about 5.3 KBytes. With this limited chunk size, large objects would have lengthy data chunk sequences in their transfer. Figure 9.4 and Table 9.2 show the average number of chunks in the chunk transfer sequence with respect to object size.

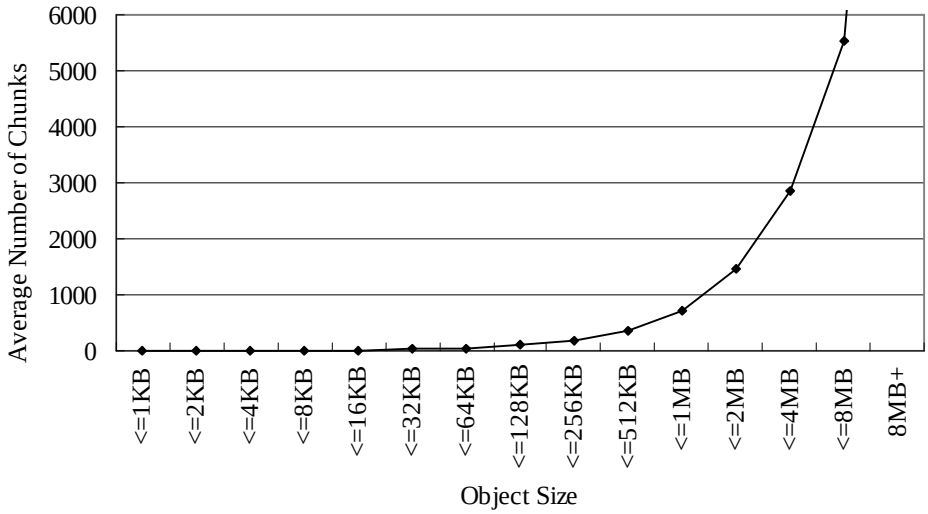


Figure 9.4 Average number of chunks w.r.t. object size

Table 9.2 Average number of chunks in object transfer w.r.t. object size

Object size range	Average number of chunks in object transfer
<=1KB	1.06
<=2KB	2.04
<=4KB	3.60
<=8KB	6.66
<=16KB	12.58
<=32KB	24.42
<=64KB	47.03
<=128KB	89.45
<=256KB	179.30
<=512KB	357.80
<=1MB	715.87
<=2MB	1469.68
<=4MB	2846.48
<=8MB	5520.78
8MB+	16561.22

From Figure 9.4 and Table 9.2, we see that the number of chunks for large objects is significantly bigger than that of small objects. It grows almost exponentially as the object size increases. When an object is larger than can fit into a small number of chunks, long Chunk Sequence Time (CST) for transferring the lengthy chunk sequence will be encountered. This long CST time will become the dominating factor for object

latency compared with other latency components, as we can see from Figure 6.5. For example, Figure 6.5 shows that for objects with size larger than 128 KBytes, the CST time occupies about 95.58% of the whole object retrieval latency on the average.

When a web page contains big objects, the retrieval latency of the big objects would become the performance bottleneck of whole page latency. To reduce the retrieval latency for large objects could effectively improve whole page latency for such pages. This is especially important as more and more significant influence is imposed on page retrieval latency by the increasing number large web objects. The concept of parallelism can be extended to chunk transfer sequence level to help with the problem. If we divide the lengthy chunk transfer sequence into multiple smaller sub-sequences and transfer them in parallel, the retrieval latency of the large object could be significantly reduced, which in turn would effectively reduce the whole page latency.

In the following sections, we propose and study the chunk-level parallelism for web retrieval.

9.2.2 Chunk-Level Parallelism (CLP)

The basic idea of *Chunk-Level Parallelism* (*CLP*) is to divide the body of a large object into multiple portions and retrieves them in parallel. This requires the ability to retrieve partial content of an object in the web system. The support for partial object from HTTP/1.1 makes this idea feasible.

HTTP/1.1 introduces a new HTTP header “Range:”, which allows clients to specify and retrieve any part of an object’s content. This feature is intended to reduce unnecessary network usage by allowing partially-retrieved data to be completed without transferring data already held by the client. It is useful in resuming broken data transfers and retrieving specific parts of objects, e.g. the descriptor fields of

multimedia files, the first a few pages of a document, and so on.

The “Range:” HTTP header specifies desired portions of objects using byte ranges. Because most of the data in the web is represented as a byte stream in practice, so they can be addressed with a byte range. The client requests a byte range via the “Range:” HTTP header. Byte range request is made like any other HTTP request, with the addition of the “Range:” HTTP request header. The parameter name “bytes” comes after this header, followed by an equal sign and the byte range specification. Below is an example web request with such “Range:” header[7]:

```
GET /Protocols/rfc2616.html HTTP/1.1
Host: www.w3.org
Range: bytes=8760-10536
Connection: close
Accept: */*
```

Each byte range of the object content is considered as a “partial object” of the original object. This partial object concept is only supported by HTTP/1.1. In the case where it is not supported, the byte range in the request will be ignored and whole object will be returned. Since HTTP/1.1 is getting its popularity, we expect to see that most web systems would already have this support.

With the support of range requests, we can implement CLP in web retrieval. Basically, the idea is to use multiple range requests to retrieve different parts of an object in parallel. The detailed process is described as follows.

When a client retrieves a web object, it first issues a normal request to the server. We refer to this request as *Master Request* in our CLP study, and the process associated with this request is referred to as *Master Retrieval Thread*. When the server serves the request and sends back the data, the first data chunk returned generally contains the HTTP headers for that object. On receiving the HTTP headers, the client will examine

[7] For details of byte range requests, please refer to RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1.

if the following conditions are satisfied:

- The server support HTTP/1.1 (i.e. it supports range request)
- There is a “Content-Length” header and the value of this header exceeds certain threshold

If the above conditions are satisfied, CLP will take place. The client will divide the object size into *k* parts based on some factors such as the “Content-Length” of the object and the bandwidth of the network etc. Then, it will trigger *k-1* new requests and assign each one of them to fetch about *1/k* of the object body (actually, the size assigned to each of these *k-1* new requests is smaller than *1/k* of the object size, we will discuss this further later). In contrast to the Master Request, these *k-1* new requests are referred to as ***Slave Requests***, and the process associated with each of them is referred to as *Slave Retrieval Thread*. After the *k-1* Slave Requests have been successfully issued, the Master Retrieval Thread will be stopped after it receives about *1/k* of the object body (in fact, the size will be larger than that). When all the parts are retrieved, they will be assembled together and the original object is got. Because the *k* retrieval processes, 1 master retrieval thread plus *k-1* slave retrieval threads, are carried out in parallel, so the retrieval speed would be much faster than one single retrieval process which fetches the whole body of the object.

The above process of CLP is depicted in Figure 9.5. We see that by dividing a large object into multiple smaller parts and retrieving them in parallel, the overall cost distance of the WRDG graph can be reduced significantly.

Note that CLP is only used to accelerate the retrieval processes of large web objects; it will not affect the correctness of web content. In the situation where partial object is not supported, the slave requests will be ignored and the master request will continue to retrieve the whole body of the object as usual without being stopped at *1/k*

of the object body.

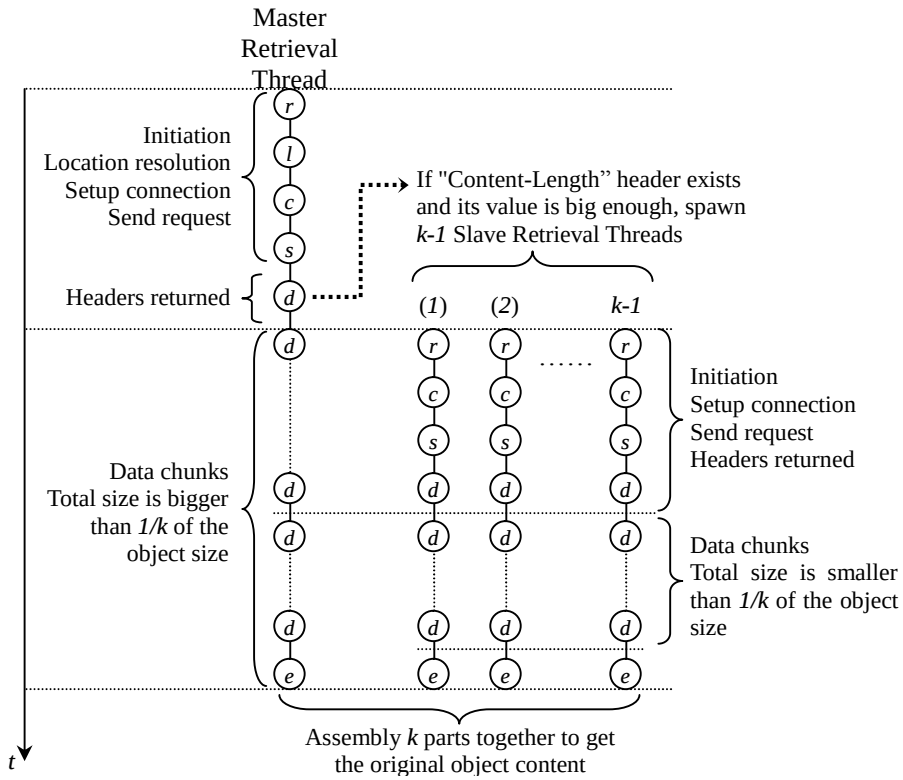


Figure 9.5 Retrieval process of chunk-level parallelism

9.2.3 Prerequisites for Chunk-Level Parallelism

There are two important questions regarding CLP to answer:

- Under what situation, should CLP be used?
- What is the threshold size for CLP to happen? How many parts should an object be divided into when it is chosen for CLP?

Many factors are involved in answering the above questions, such as object size, network bandwidth, number of connections, and connection time etc. Below we will discuss these factors and answer the above questions.

First, let us look at the first question: Under what situation, should CLP be used?

As stated earlier, CLP is proposed for the retrieval of large web objects. CLP can only take place when the size of the object is available and it is greater than a threshold size. The threshold size is determined by multiple factors such as the connection setup

time. For example, if the connection setup time is long, the threshold size for triggering CLP has to be big. Otherwise, there may not be any benefit for doing CLP since the Master Retrieval Thread could have already retrieved the whole object body before the Slave Retrieval Threads have yet finishing setting up the connections.

Because CLP requires dealing with partial content of objects, so it only works with HTTP/1.1-compatible web systems since the partial object concept is mainly supported by HTTP/1.1 currently. In the situation where partial object is not supported, CLP should not be considered.

Since CLP tries to divide a large object into multiple parts and retrieve them in parallel, it will increase the number of concurrent network connections used in a web system. This requirement may be difficult to satisfy for extremely busy web servers. However, we would like to point out that:

1) Most web servers are not extremely busy in most of the time. So, CLP should have very little problem to work with them.
2) For those extremely busy web servers, the problem can be alleviated with the help of CDN. If a busy server subscribe to CDN service, the requests to it will be re-directed to different servers transparently. In this situation, the multiple concurrent network connections established by CLP will be distributed to different servers, and each server will get only a small portion of the connections. Therefore, the demand on concurrent network connection on a server will not be high. Since extremely busy web servers tend to have already subscribed to CDN service (otherwise, they should think of doing so since they are extremely busy), so we expect CLP would also work well with such servers.

Network bandwidth also has influence on CLP. Because CLP actually tries to accelerate the retrieval process of large web objects by utilizing spare network

bandwidth for multiple concurrent web requests, so, it will perform better if there is more spare network bandwidth to use. For the environments with very limited network bandwidth, CLP may not be a good choice for web acceleration.

Next, we move on to answer the second question: What is the proper threshold size for CLP to happen? How many parts should an object be divided into when it is chosen for CLP?

First, let us decide the number of parts that an object should be divided into for CLP. The number of parts is tied to the size of each part for a given object. Basically, the more parts an object is divided into, the smaller each part would be, and the shorter the retrieval latency of each part would be. However, to divide an object into too small parts for CLP may not help much in reducing whole page latency because of two reasons:

1) If we divide an object into too small parts in CLP, the size of each part could become smaller than other objects in the page. In that case, the page retrieval latency would be dominated by other objects. This suggests that it may not be helpful to divide an object into parts smaller than other objects in the page.
2) Web retrieval process typically undergoes long connection setup time and TCP slow-start effect, which make the retrieval latency for smaller objects often comparable to that of bigger objects (refer to Chapter 6 for details). This suggests that to divide an object into too small parts is not cost-effective in terms of retrieval latency and resources used.

Taking into consideration the above factors, we would recommend that a large object should be divided into *k* parts so that the size of each part is around the average size of most commonly seen web objects. From Chapter 6, we know that the average size of objects is about 5.71 KBytes. We would suggest using this size as the size of partial

objects for CLP mechanism, especially in our environment.

On the other hand, dividing a large object into many small parts and retrieving them in parallel would impose extra demand on concurrent network connections and server load. When an object is very big, the number of the parts could be quite large. Then the burden on network connection and server load could be excessive, which could have negative effects on the performance of web retrieval. To refrain this from happening, we set a cap value N for k (which is the number of parts that an object is divided into), i.e.:

$$k \leq N$$

In our study, we vary the value of N and investigate the effect of it on the performance of CLP. The values of N we used in our study is: 2, 4, 8, 16 and 32.

In summary, we use the following method to decide the number of parts that an object should be divided into for CLP. First, we assign the size of each partial object to be the average web object size (5.71 KBytes), and we calculate the initial k by using this size to divide the object size. If k is not greater than N, then we will use this k and this partial object size to do CLP for that object. If k is greater than N, we will set k to N and use this k to calculate the size of each partial object. Then we use these values of k and partial object size for CLP. In any case, k will not be greater than N, and the size of each partial object will always be greater than or equal to the average web object size. This way, we have avoided imposing excessive demand on network connection and server load, while still attained the effectiveness of CLP.

Now let us deduce the threshold size for CLP. As we showed in Chapter 4 and Chapter 6, every web request would have 5 latency components, i.e. LRT, CT, RST, CST and ET. In CLP, the slave requests would also have these latency components except the LRT component because slave requests can directly get the server address

from the master request. So, the slave requests would undergo the latency components CT and RST while the master retrieval thread is receiving object data, this put certain constraints on the proper object size for CLP.

Figure 9.6 shows the relationship between latency components and size ranges in CLP. In the graph, time runs down the page. The vertices that represent the operations in the retrieval processes are omitted. Instead, we put the names of the latency components beside the retrieval process lines to depict the existence of those operations. Those large braces indicate some relations of the timings and sizes.

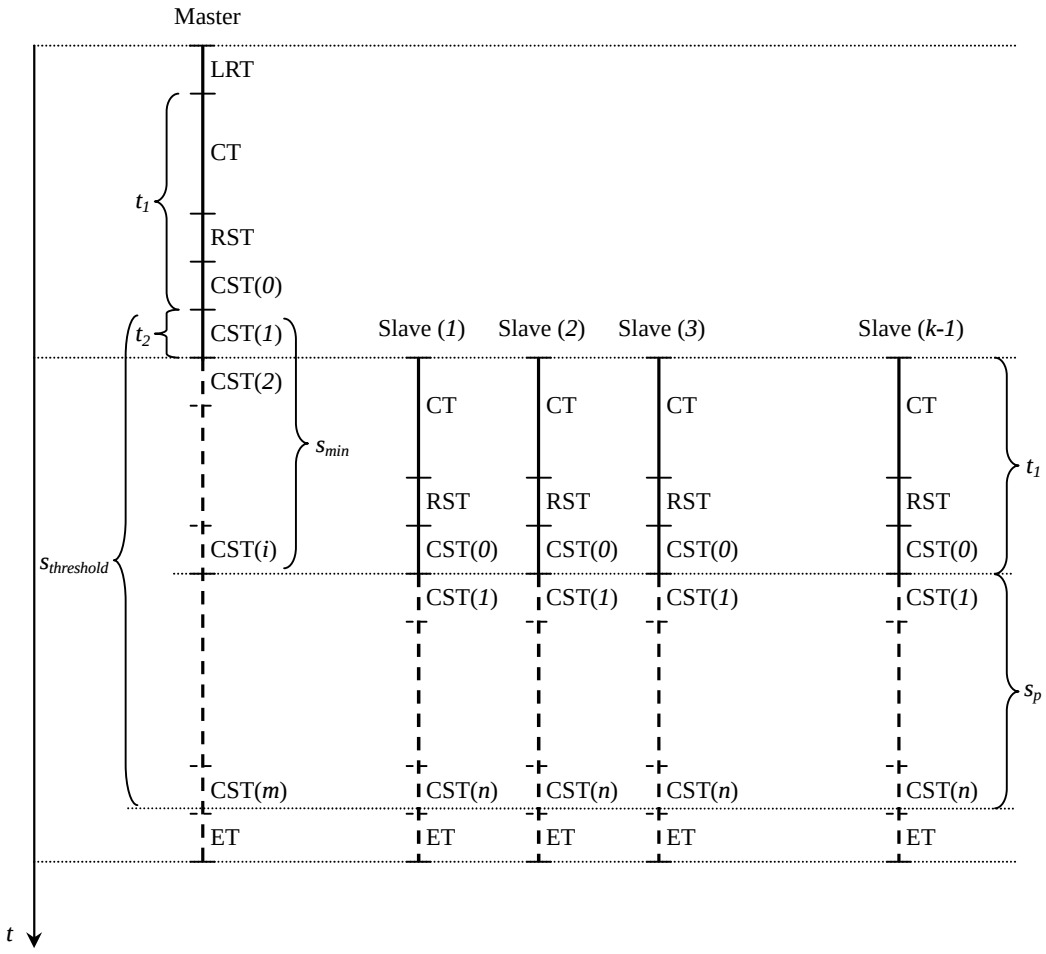


Figure 9.6 Relationship between latency components and size ranges in chunk-level parallelism

From Figure 9.6, we can see clearly that the master retrieval thread would have already received s_{min} bytes of the object data when the slave requests finish the latency

components CT+RST+CST(0) only. It is obvious that the object size must be considerably bigger than s_{min} if we want to do CLP. So, we must take factors like this into consideration when deciding the proper object size for doing CLP. Below we will try to deduce the formula for calculating the proper threshold size.

Because many characteristics about web retrieval vary greatly, e.g. the latency components and chunk size fluctuate considerably due to the status of network and workload on server etc, so it is rather difficult to produce an accurate formula for CLP. But we can develop a rough model for the relationship among the factors based on the following assumptions:

1) No persistent connection is used in CLP, and the connection time is constant for all retrieval processes

   This is to assume that every slave request in CLP will undergo the latency component CT and this component is constant for all of them. This assumption is somewhat reasonable because persistent connect is not well supported even in today's web system. On the other hand, our formula/deduction can be extended to handle persistent connection as well. To make the deduction uniform and simple, here we just assume there is no persistent connection in CLP and the connection time is constant.

2) All slave retrieval threads have the same CT+RST+CST(0)

   We use t_1 to represent this latency, i.e.

   t_1 = CT+RST+CST(0)

3) The first chunk contains only HTTP headers

   In our experiments, we found that this statement is true for most cases. Nevertheless, even if the first chunk also contains object body data, the amount of the data is often negligible compared with the whole object body. Moreover, our

formula/deduction can also be extended to handle the situations where the first chunk also contains object data, or HTTP headers are found in more than one chunk.

4) The size of all chunks are the same

In reality, chunk size varies considerably. For simplicity reason, here we use the statistical average of the chunk size as the size for every chunk. According to our study in Chapter 6, the average chunk size in web retrieval is about 5.3 Kbytes. We use s_{chk} to denote the size of one chunk. So we have:

s_{chk} = 5.3 Kbytes

5) The latency for every chunk is the same

We assume that the transfer latency for every chunk is the same, and we use *CST(i)* to denote the transfer time for *Chk(i)*. Because it is difficult to obtain the latency of the first chunk *Chk(0)* and we assume *Chk(0)* contains only HTTP headers, so we use the latency of the second chunk *CST(1)* as the unit latency for transferring one data chunk. We denote this unit latency of one chunk as t_{chk} . From Figure 9.6, we have:

$t_{chk} = CST(1) = t_2$

6) All slave retrieval threads are started simultaneously, i.e., the starting time of all slave requests are the same.

7) The partial object size s_p assigned to each slave request is the same, and we use the average size of web objects (5.71 KBytes) as the minimum size for each partial object, i.e., we have:

$s_p \geqslant$ 5.71 KBytes (F9.1)

Based on the above assumption and Figure 9.6, we see that the object size satisfies the

following equation:

$$s_{object} = s_{min} + \quad k \times s_p \quad (k \geq 2) \tag{F9.2}$$

For CLP to happen, k must be greater than or equal to 2. Substituting k with the minimum value of 2 in the equation (F9.2) will give the minimum object size required for CLP, i.e. the threshold size $s_{threshold}$:

$$s_{threshold} = s_{min} + 2 \times s_p \tag{F9.3}$$

For s_{min} , we have the following equation (see Figure 9.6):

$$s_{min} = i \times s_{chk} \tag{F9.4}$$

From Figure 9.6, we know that $t_1 = (i - 1) \times t_{chk}$. So,

$$i = \frac{t_1}{t_{chk}} + 1 \tag{F9.5}$$

Integrate the equation F9.5, F9.4 and F9.1 into F9.3, we will get:

$$s_{threshold} = \left(\frac{t_1}{t_{chk}} + 1 \right) \times s_{chk} + 2 \times s_p$$

$$s_{threshold} \geqslant \frac{t_1}{t_{chk}} \times 5.3 + 16.72 \quad \text{KBytes} \tag{F9.6}$$

In equation F9.6, both t_1 and t_{chk} can be obtained by real-time monitoring the retrieval process (note that we assume $t_{chk} = CST(1) = t_2$). Therefore, $s_{threshold}$ will be able to be obtained during the retrieval process.

With the above knowledge, we are now able to give a more detailed description of our CLP mechanism. Figure 9.7 gives the process flow of this mechanism. By monitoring the retrieval process, web client will be able to record the t_1 and t_{chk} , and calculate $s_{threshold}$ in real-time for each large object request. Note that we use the latency of the second chunk, i.e. *CST(1)*, as t_{chk} . So, all slave requests are triggered after the second chunk in the master retrieval process has been returned from the server.

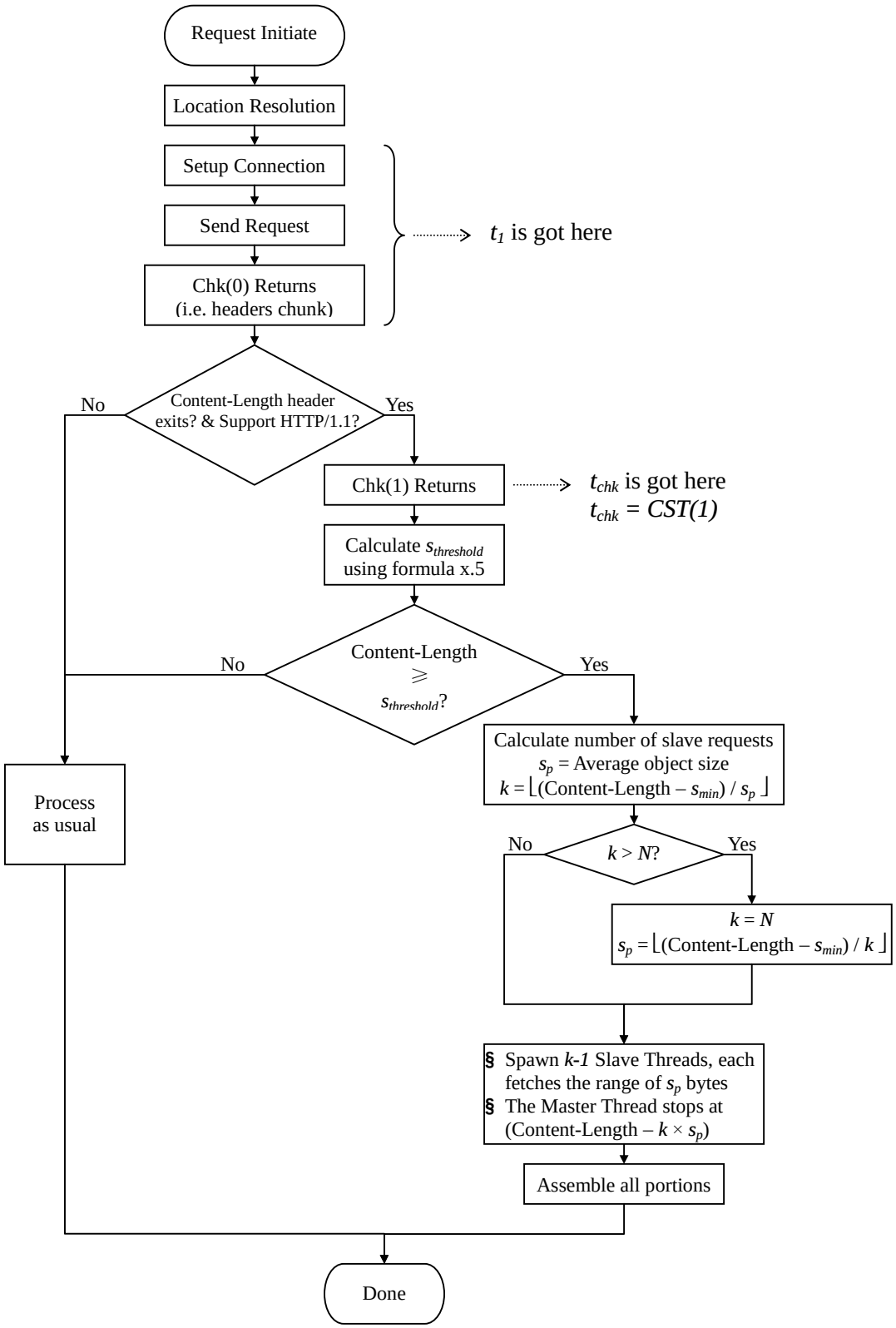


Figure 9.7 Process flow of chunk-level parallelism

After slave requests have been successfully triggered, the master retrieval process should be stopped prematurely when the size of the data it receives has reached the value assigned to it.

9.3 Performance Study

To study the performance of CLP, we conducted both simulation experiments and real system tests. For the simulations, we used our detailed chunk-level traces described in Chapter 5. For the real system, we have implemented a working system based on Squid 2.4.STABLE3 to perform the CLP task. The system will monitor the conditions (see previous section) and do CLP for those object requests which satisfy the conditions. Other aspects of the testing environment are as described in Chapter 5. Below we report our results from these simulations and real system tests on CLP.

From the formula F9.6 we can see that the time t_1 and t_{chk} play important roles in CLP. So we would like to have some study on them first. Our CLP system bases on the ratio of t_1/t_{chk} to calculate the suitable threshold size $s_{threshold}$ for triggering CLP. Figure 9.8 plots the distribution of the ratio of t_1/t_{chk} .

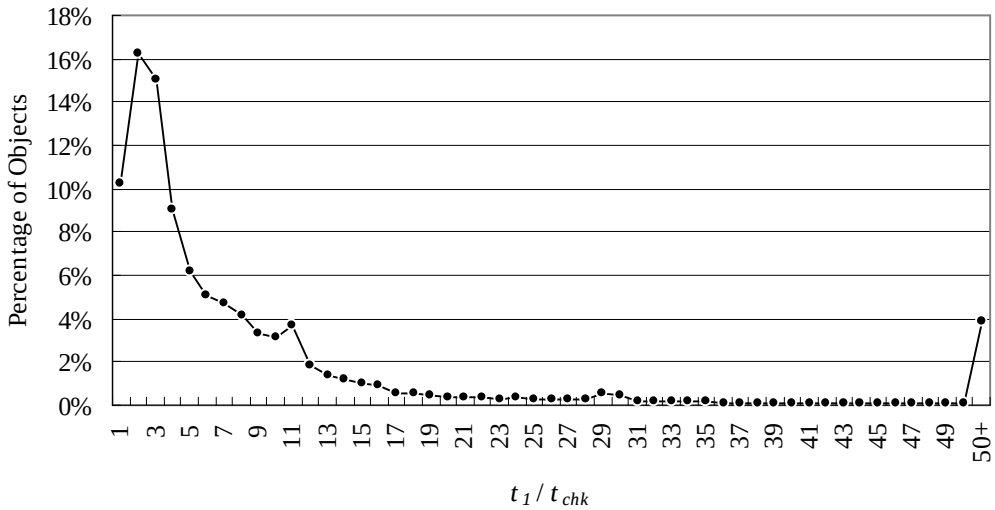


Figure 9.8 Distribution of the ratio of t_1/t_{chk}

From the graph, we see that the values below 15 take up the major portion (more

than 85%) of the distribution of t_1/t_{chk} . However, there exists much bigger values for the ratio of t_1/t_{chk} , from 50 to up to more than 10,000. Those big values contribute nearly 4% to the overall distribution of t_1/t_{chk} . This put the average value of t_1/t_{chk} to be at about 26.3. In other words, t_1 is about 26.3 times bigger than t_{chk} on the average. This result shows that t_1 is surprisingly big as compared with t_{chk} . The reason for this is because t_1 contains the connection time CT, which is very big in web retrieval. Recall in Chapter 6, our study showed that CT is one of the most significant latency components in object retrieval. It takes up from 3.7% to 77% of the object retrieval latency (refer to Figure 6.5).

The above observation is very important because it implies that CLP is not suitable for medium-sized objects as every retrieval thread will undergo at least t_1 , which already equals to the transfer time of many chunks. Using formula F9.6, we can get the threshold size for CLP:

$s_{threshold} \geqslant$ 156.11 KBytes

This result indicates that on average, an object should be larger than 156 KBytes to be suitable for CLP to take place. This size shall be considered quite big in current web system. In our trace, we only observed about 3.31% web pages have objects larger than 156 KBytes. Although the percentage is not very big, we believe it could go higher in the future as large digital material objects are getting more and more popular on the web. Nevertheless, 3.31% is still significant enough for us to look into effective mechanisms to accelerate the retrieval process of them.

Figure 9.9 studies the performance of CLP on retrieval latency of individual objects. Here we set N to be 8 (N is the cap value for k, which is the number of parts that an object is divided into).

From this graph, we see that the effect of CLP on object retrieval latency is

substantial. It can reduce the retrieval latency of large objects dramatically. The simulation results show that the improvement ranges from 77% to 87%, with an average of 83.86%, while the real system testing achieves 68% to 86% improvement, with an average of 80.6%. In general, the improvement gets better as the object size increases. This is expected as larger objects have lengthier chunk sequences, which can be effectively improved by CLP.

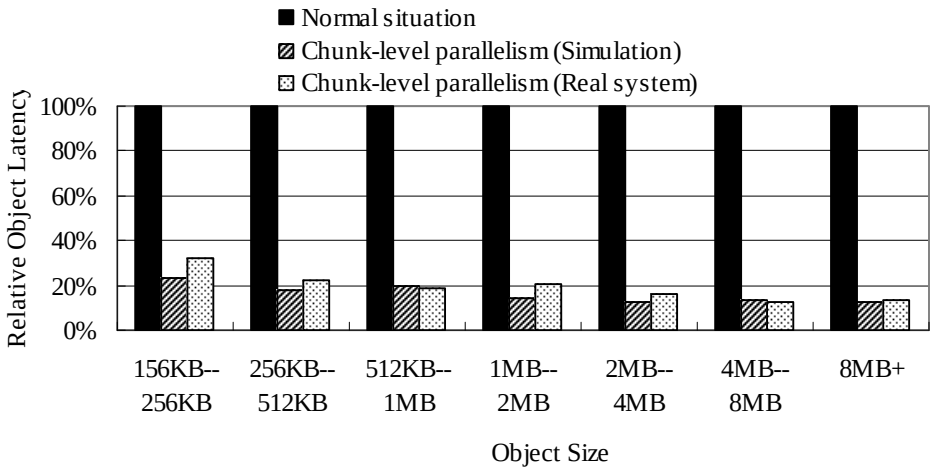


Figure 9.9 Effect of chunk-level parallelism on retrieval latency of individual objects

Note that the performance of real system is often lower than the simulation results. This could be due to the fluctuation of network and server status, which are largely ignored in simulations.

Figure 9.10 plots the effect of CLP on retrieval latency of pages containing large objects. In simulations, CLP improves page retrieval latency from 38% to 85%, with an average of 68.6%. In real system tests, the improvement ranges from 27% to 84%, with an average of 64.5%. Again, we see that the effect of CLP on page retrieval latency is also very significant, and the improvement generally gets better as the size of the largest object in the page increases. This is actually understandable because: For pages containing large objects, the whole page latency will be dominated by those large objects; since CLP can effectively reduce the retrieval latency of large objects,

this reduction will inevitably be reflected on whole page latency.

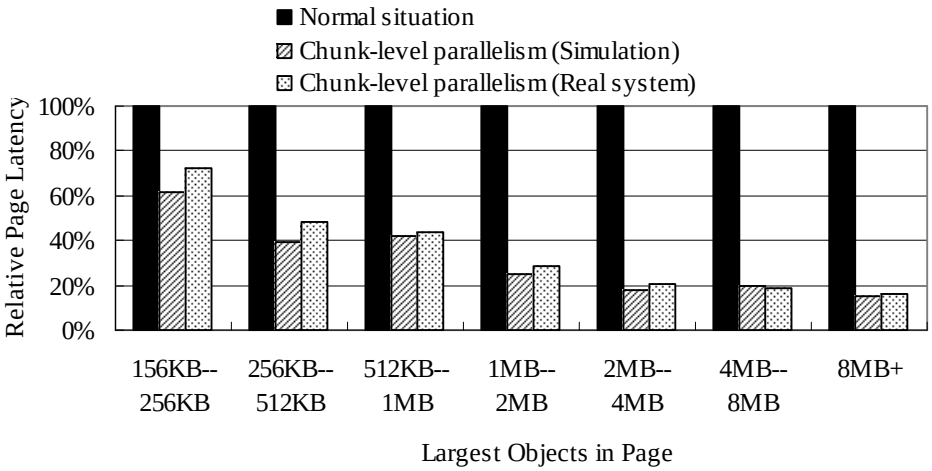


Figure 9.10 Effect of chunk-level parallelism on page retrieval latency

We noticed that the improvement that CLP achieves on page latency is generally lower than that on individual object latency. This could be due to the following reason:

When large objects in a page are divided into smaller parts by CLP, the retrieval latencies of other objects in the page may become more dominating to whole page latency. In other words, there are some other latency contributors that prevent CLP's improvement on individual object latency from being fully reflected on whole page latency.

However, when the objects that CLP works on are extremely large (e.g. bigger than 8 MBytes), the improvement on page latency would be close to the improvement on individual object latency. This is because: when page contains object with extremely big objects, the retrieval latency of other objects become very negligible and the page latency is almost solely determined by the extremely big objects even after they have been divided into smaller parts. So, in this situation, the improvement on the extremely big objects will be almost fully mapped into the improvement on whole page latency.

Figure 9.11 studies the effect of N on the performance of CLP in terms of page

retrieval latency. Here *N* is the cap value of *k*, which is the number of parts that an object can be divided into.

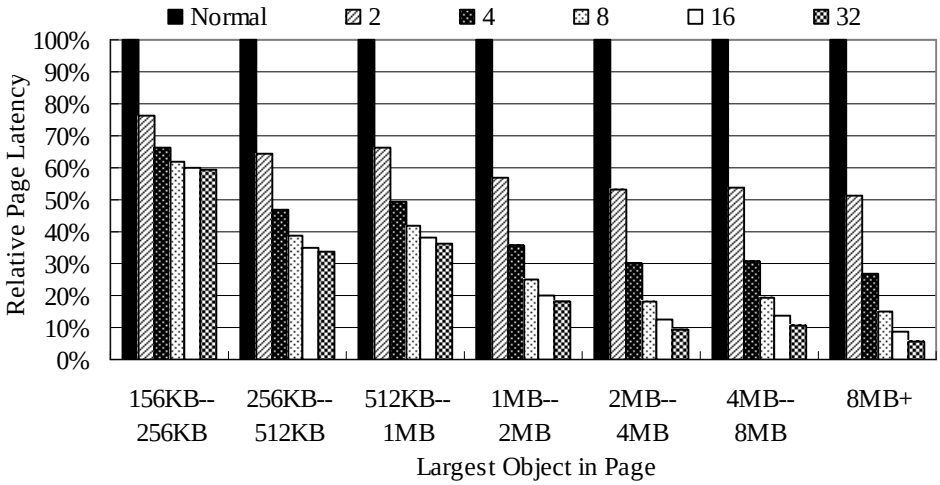


Figure 9.11 Effect of N on the performance of chunk-level parallelism

From the graph, we can see that the performance of CLP generally increases as N increases. However, the relative increase for different N becomes less significant for big values of N. This is more obvious for moderate large objects such as those between 156 to 256 KBytes. This suggests that different values of N should be considered for different sizes in order to achieve the best cost-effectiveness. For moderate large objects, 4 or 8 may be suitable values for N. But for very large objects, much wider parallelism width (e.g. 32) should be considered.

We also noticed that the performance gain is far from being directly proportional to the increase in N for pages with relatively smaller objects (e.g. 156 to 256 KBytes objects), while it nearly has the directly proportional relationship for pages with very large objects. This could be due to the same reason stated earlier on: For pages with relatively smaller objects, other objects in the same page could prevent CLP's effect from being fully reflected on whole page latency. But for pages with very large objects, this phenomenon will not be seen since the very large objects would always dominate whole page latency even after they have been divided into smaller parts.

Because CLP divides large objects into multiple smaller parts and retrieve them in parallel, it will impose extra demand on concurrent network connections and server load. If such extra demand is too excessive, it could have negative effects on the overall performance of web system.

However, in our study, we found only about 3.31% web pages satisfying the conditions for doing CLP. The number of the large objects (i.e. larger than 156 KBytes) contained in those pages only count for about 0.2% of the total objects in all pages. So the extra requests and connections created by CLP on these large objects would only be (N−1)×0.2%. When N is 8, there will be only 1.4% extra requests and connections introduced by CLP. This extra demand shall be considered as very small. Therefore, we expect the overhead that CLP introduces to be marginal in current web system.

9.4 System Implementation Considerations

To examine the performance of CLP in real environment, we have implemented a working CLP system based on Squid 2.4.STABLE3. The system works as a proxy between clients and servers. While serving requests, our CLP proxy system will monitor the conditions (see previous section) for CLP. Once the system finds a request satisfying the conditions, it will do CLP for that request. We choose to implement CLP on a proxy system rather than a web client program because of two reasons: Firstly, to implement a real CLP system, we need to work on the source code of the system. Many web client programs like MS-IE and Netscape do not give access to their source codes. So we choose to work on the open source system Squid. Secondly, if we implement the CLP capability into a proxy system, it can later be used for all types of web client programs such as MS-IE, Netscape and wget [284] etc. to enable them to take the benefit of CLP as well. Moreover, implementing CLP in a proxy cache system would also enable web accesses to take the advantage of web caching.

Unlike simulation experiments which are easy to implement, real system with CLP capability is rather difficult to implement because there are a number of complicated factors to be taken into consideration. Below we discuss some design issues we addressed during the implementation of our CLP system.

Recording the times

Timing plays an important role in CLP. In formula F9.6, we see that CLP rely on t_1 and t_{chk} to calculate the threshold size $s_{threshold}$ for CLP. Because t_1 consists of the latency components CT, RST and CST(0), so we should start recording the time for t_1 from the point when the system is trying to setup the network connection. In Squid, this can be done in the function `fwdConnectStart()`. The timing for t_1 ends after the first chunk has returned. This happens in the function `httpReadReply()`. As for t_{chk}, it shall also be recorded in the function `httpReadReply()` since all the replies from servers are handled by this function.

The t_1 and t_{chk} for different requests could vary greatly as the requests may go to different servers at different locations. So the recorded times should be kept with each request. This can be achieved by adding new fields in the data structure `clientHttpRequest` or `request_t` and storing the timings in them. This way, the timings would be always ready for use for each request.

Checking the conditions for CLP

As we know in Section 9.3, CLP would take place only when certain conditions are satisfied. Most of the conditions are related to HTTP response headers. In Squid, HTTP replies (both the HTTP headers and the object body data) are handled by the function `httpReadReply()`. We modified this function to let it check: (1) if the server supports partial content retrieval, i.e. if it supports HTTP/1.1; (2) if the status code of the reply is OK; and (3) if the “Content-Length” header exists. By examining

the status line of the headers, the system would know the status code and whether the server is HTTP/1.1 compliant. If the system finds a request satisfying these three conditions, it will get t_{chk} (recall that we use the latency of the second chunk as t_{chk}) to calculate the threshold size $s_{threshold}$ for this request. If the value of the "Content-Length" header is greater than this $s_{threshold}$, the system will try to do CLP for it by calling new functions added by us to spawn slave requests.

Note that t_1 and t_{chk} can sometimes fluctuate greatly due to the variation in network status and server load, this could make the calculated threshold size $s_{threshold}$ unrealistic, i.e. extremely small or large. To adjust this deviation, we also set a global minimum threshold size (e.g. 128 KBytes). If the calculated threshold size is unrealistic, we will use this global minimum threshold size to compare with the "Content-Length".

There is also another important point we need to be paid attention to. In the case where the first chunk also contains object body data besides the HTTP headers, the amount of the object body data in this first chunk should be counted in the size assigned to the master retrieval thread, if CLP happens for this request.

Spawning slave requests

The new functions for spawning slave requests will first compute the size ranges to be assigned to each slave request. Then they will create the request messages and network connections for all the slave requests. Typically, the request messages for the slave requests should have the same headers as the original master request, except that they also have the "Range:" header. For setting up network connections, the functions will first check if there are persistent connections available in the system. If there is, the CLP system will make use of them. Otherwise, it will open new connections for the slave requests as Squid usually does for normal requests, i.e. by calling the function

`commConnectStart()` to do it. When connections setup is done, request messages for those slave requests will be sent out through them.

There is certain management data associated with each slave request, e.g. the size range assigned to it, the size of received data, and the received data etc. Each set of such data should go with each specific slave request. To do this, we add new fields to the data structure `FwdState` and create one such structure for every slave request. Note that each slave request has its own memory buffer for storing the received data.

Receiving the partial data

The master retrieval thread receives object data in the function `httpReadReply()` as usually, only that it should be forced to stop when it finishes the size assigned to it.

Because the data the master retrieval thread receives belongs to the first part of the object body, so it can be sent it to client immediately. However, for slave retrieval threads, the data they receive may not be able to be sent to client immediately because the order of the received ranges of data may be out of sequence. But whenever a slave retrieval thread receives a chunk of data, we will have it to check whether the data can be sent to client, i.e. whether the data range before this chunk has already been sent to client. As long as the system finds that the current chunk of data is in order with those which have been sent to client, it will send the data from slave retrieval threads to client immediately, without waiting for the whole threads to finish. At the same time, the data will be merged to the master retrieval thread to recover the whole object for caching.

Finishing CLP requests

When all the master and slave retrieval threads finish receiving data, the resources occupied by them will be released. Such resources include memory buffers,

data structures, file descriptors, and network connections, etc. For the network connections, they can be put in the persistent connection pool for future use, or freed immediately. In our system, this is configurable in the configuration file `squid.conf`.

Note that the whole object data retrieved by master and slave retrieval threads are already sent to client prior to the release of the resources taking place. The whole object will also be submitted for caching when the CLP finishes. Since all the partial portions have been assembled together to recover the whole object before it is submitted for caching, so we do not have the issue of caching of partial content in our CLP system.

Avoiding Resource contention

Our CLP system is implemented on a web proxy system. As web proxy can be very busy sometimes, the system may run short of resources in some extreme situations. When such situation happens, we should give normal web requests higher priority over the slave requests spawned by CLP in using the resources. Our CLP system will monitor the usage status of the resources (such as the number of connections). If it detects that the usage of certain resources reaches a threshold point, it will suppress CLP to certain degree, up to zero. By doing so, our CLP system could assure the quality of service for normal web requests while still take the benefit of CLP when situation permits.

9.5 Conclusion

In this chapter, we exploited fine-grained parallelism for the acceleration of web retrieval. By extending the concept of parallelism to intra-object level, we proposed the Chunk-Level Parallelism (CLP) mechanism to improve web retrieval performance for large objects. Our comprehensive study on CLP revealed some important relations

regarding chunk-level parallelism such as the proper threshold size for CLP to take place. By selecting proper parameters for CLP based on the relations, we have attained high effectiveness of CLP while avoided imposing excessive demand on network connection and server load. We conducted simulation experiments as well as real system tests to study the performance of CLP. Our results show that CLP can achieve about 83.86% and 68.6% improvement on object retrieval latency and whole page latency respectively when large objects are in presence. As more and more large digital documents, multimedia materials and web-based applications etc. are increasingly distributed over the web, the CLP mechanism could become more effective and preferable in the future.

Chapter 10 Conclusions

10.1 Summary

This thesis addressed the issues in the area of modeling and acceleration of web content delivery. In the thesis, I first examined the traditional way of web acceleration, i.e. caching-based mechanisms. By investigate the factors affecting the cacheability of objects and their utilization in current web system, I found that the cacheability of objects is not well utilized due to the absence or improper value of critical HTTP headers from web servers. If current web servers can be configured more properly to provide directives for better cacheability, considerable improvement can still be brought to the performance of caching-based mechanisms.

I proposed a fine grained Web Retrieval Dependency Model (WRDM) in this thesis to address the issue of lack of precise model for studying web retrieval latency. Our detailed study on web retrieval based on WRDM model shed light on the details of web retrieval latency. It revealed that the relationship between object latency and page latency is very complicated and the actual object fetch latency is often less of a problem for web retrieval than Definition Time and Waiting Time when page latency is concerned. Using the WRDM model, I also analyzed the possible impact of real-time content transformation on web retrieval latency and derived various upper bounds for web acceleration, which revealed low-level impacts of real-time content transformation and potentials of web acceleration.

With the guidance of the WRDM model, I analyzed the effect of an important acceleration mechanism, namely web compression, through low level studies. The detailed analysis brought us insights of some important effect and implication of compression on page retrieval latency.

Realizing the deficiency of general-purpose compression algorithms in the

specific area of web content delivery, I proposed a new compression mechanism, named Content-Aware Global Static Compression (CAGSC), to improve the performance of compression in web content delivery.

Based on the findings from the studies using our WRDM model, I proposed new ways to web acceleration. Besides the novel compression mechanism mentioned above, I also proposed and studied innovative acceleration mechanisms in two aspects, i.e. dependency related mechanisms which are the Server Location Propagation mechanism (SLP) and Embedded Object Information Propagation mechanism (EOIP), and parallelism related mechanism Chunk-Level Parallelism (CLP). Experimental results showed that these mechanisms can produce considerable improvement on web retrieval latency.

10.2 Contributions

This thesis mainly focuses on the area of acceleration of web content delivery. I introduced an innovative fine grained model and proposed new ways to web acceleration. The main contributions of this thesis are listed as follows:

- Systematic study on the cacheability of objects in current web system

  I studied the performance of web caching by systematically investigating how web caching mechanism works from the internal of a real caching system, and how well those essential cacheability-controlling HTTP headers are presented in current web system. I dug into the relationship among the co-occurrent factors and revealed the effectiveness of the factors in the multi-factor co-occurrent situation. The study revealed the effective factors and proper settings for TTL. By improving them, the performance of web caching can still be improved considerably.

- Proposed a new precise model for studying web retrieval latency

  Most existing studies on web retrieval are based on object level information.

Knowing its limitation, I proposed a detailed operation and chunk level Web Retrieval Dependency Model (WRDM) to provide more precise capture of web retrieval. This model helps us to understand the root causes of the latencies for both individual objects and whole pages. It can also act as an effective tool in developing and analyzing web acceleration mechanisms.

- Chunk level study on object retrieval latency

  I conducted detailed study at operation and chunk level on object retrieval latency. While the results re-confirmed the large contribution of CT and CST to object latency, I also made some other important findings. I found that the retrieval latency for smaller objects is often comparable to that of bigger objects for the group of objects with size smaller than 4 KBytes. Another important finding is that the latencies for chunks with different sizes are quite randomly distributed, which indicates that mechanisms which aim to reduce chunk size may not help much in reducing object retrieval latency.

- In-depth understanding and study of the factors affecting page retrieval latency

  Our detailed study based on the WRDM model revealed complex factors affecting page retrieval latency, which confirms our argument that the mapping from object retrieval latency to page retrieval latency is very complicated. When objects are put together to form pages, their actual fetching latency become less significant in determining page retrieval latency. Instead, two new latency components particularly found in pages, i.e. DT time and WT time, become the dominating factors. I thoroughly studied the relationship among the number of objects in a page, DT and the dependency between objects in a page, WT and the parallelism in web retrieval. Our results revealed the effect of these factors on page retrieval latency and the complex inter-relationship among them. In order to achieve high

performance of page retrieval, we need to take all the factors into consideration simultaneously. Simply considering one of them will not yield the best improvement because other factors will soon become the performance bottleneck if only one is improved.

- Revealed the impacts of real-time content transformation on web retrieval latency

  Web content transformation has been an important technology to satisfy the different expectation of web users. There are many studies focusing on the real-time feature and the restrictions on the kind of transformation that can take place etc. But there is little study on the possible impacts of different content transformation approaches on web retrieval latency. Using our WRDM model, I analyzed the performance impacts of content transformation. Our results suggest that the partial-object buffering content transformation should be the preferred approach since it has little restrictions on the kind of transformation that can take place while it imposes moderate negative effect on page retrieval latency.

- Derived upper bounds for the performance of acceleration mechanisms

  I also derived various upper bounds on the performance improvement for acceleration mechanisms in this thesis. While many mechanisms have been proposed and shown promising potential of acceleration, it remains to be seen the quantitative upper bound of them. Based on the understanding of object retrieval, page retrieval and the relationship between them revealed under our WRDM model, I derived upper bounds for acceleration mechanisms, which help us to understand the potentials of web acceleration.

- In-depth analysis of web compression at chunk level

  I analyzed an important web acceleration mechanism, namely web compression. The detailed chunk level study based on our WRDM model revealed that reducing

the number of chunks is more effective in improving retrieval latency than reducing the size of every chunk. So, pre-compression almost always outperforms real-time compression since it reduces the number of chunks while the latter tends to reduce the size of every chunk. Our study also investigated the impact of compression on the DT times of EOs in a page and the demand on parallelism. The results revealed some special effect and implication of compression on page retrieval latency.

Ÿ Proposed a new compression algorithm

I also proposed a novel compression algorithm, namely Content-Aware Global Static Compression (CAGSC). The algorithm is specifically designed for web content to improve the effectiveness of compression in web content delivery. Results showed that improvements of up to 20% on object retrieval latency and 14.6% on page retrieval latency can be achieved by the new algorithm.

Ÿ Proposed new mechanisms to address the dependency introduced latency

The analysis on web retrieval based on our WRDM model revealed that there are dependencies between objects and between operations in retrieval process; and such dependencies introduce significant latency to web retrieval. I proposed innovative ways to web acceleration by manipulating such dependencies through information backward propagation. Two actual mechanisms are studied. One is the Server Location Propagation (SLP) mechanism for reducing the latency incurred by the dependency on server location resolution. The other mechanism is the Embedded Object Information Propagation (EOIP) mechanism, which aims to reduce the latency introduced by the dependency between CO and EOs. Our experimental results showed that these two mechanisms could improve whole page retrieval latency by about 4.22% and 10.66% repectively.

Ÿ Proposed new mechanism to exploit fine-grained parallelism for web acceleration

I also proposed the Chunk-Level Parallelism (CLP) mechanism by extending the concept of parallelism to chunk transfer sequence level to accelerate the retrieval process for web pages containing large objects. Both simulation experiments and real system tests showed that CLP can achieve substantial improvement of over 60% on whole page retrieval latency when large objects are in presence. This mechanism could become more effective and preferable in the future as more and more large digital documents, multimedia materials and web-based applications etc. are increasingly distributed over the web.

10.3 Future Work

The continued exponential growth of the World Wide Web not only makes it the prevailing media platform, but it also puts new challenges and higher requirement on the speed of delivery of information to users. As environment evolves with the rapid growth of the web, new directions and mechanisms need to be researched in order to provide high-quality web content delivery performance. Below we list some possible directions and works that can be performed to make further contributions to this area.

Ÿ Enhancement to web caching

Although web caching has its limitations, it remains to be an important and effective solution to web acceleration. Most current web caching systems work on static whole objects. However, the characteristics of web content have changed remarkably in recent times, which makes it inadequate to just handle static whole objects. Nowadays, more and more web content is generated dynamically. Current web caching systems lack the capability of handling such dynamic web content properly. Moreover, partial objects are also seen in current web requests, which are not well handled by current web caching systems either. So, to enhance web caching systems to make them to appropriately handle dynamic objects and partial

objects becomes an important direction in improving web retrieval performance.

- More studies focused on page latency, not object latency

  As we showed in this thesis, page retrieval latency is more meaningful to web users than object retrieval latency, and the mapping relationship between object latency and page latency is very complex. Being aware of this, it would be valuable to further study issues on page retrieval latency. Further investigation on the existing acceleration mechanisms with special emphasis on page latency would be also beneficial in understanding and improving web retrieval latency.

- Protocol and language support for information propagation

  In this thesis, we demonstrated that the information dependency in web retrieval introduces significant latency, and showed that information backward propagation can effectively manipulate such dependencies and reduce the relevant latency. However, current web protocols and languages do not support information propagation. It is necessary to further study all aspects of information propagation and to push for proper support for it from web protocols and languages.

- Developing better protocols for web content delivery

  The current web systems run on HTTP over TCP. Although these two protocols have been working well, there are many performance issues against them. Initially, HTTP opens a separate network connection for each and every object in a page, which is proven to be very inefficient. While this situation is somewhat improved in HTTP/1.1, there still remains many problems. TCP typically has a “slow-start” phase with each new connection, which reduces throughput at the beginning of each connection. TCP’s congestion avoidance mechanism makes the effect even worse. Furthermore, TCP is strictly ordered in the way it delivers packets, which could introduce considerable delay when packet loss occurs. If we could develop

better protocols for web content delivery, faster delivery speed may be expected. While there are already a few attempts in this direction such as SCTP, BEEP, and HSTP etc. [241, 242, 238], we expect more work to be done in the future.

- Application acceleration

  In recent times, web-browse-based distributed computing is getting more and more interests, and enterprises start to decentralize and move their key corporate applications onto the web. As applications are increasingly distributed over the web and more complex forms of information exchanged, there emerges a persistent problem: web applications are bandwidth and CPU hogs. Today, the problem is typically handled by installing more application servers. However, this approach is not cost-effective and its management will become extremely difficult when the number of application servers is big. A better answer to this question might be application acceleration. Although there are difficulties in doing application acceleration currently, this direction already shows promising potential in easing the heavy loads on servers and increasing the speed at which the information can be served. The benefit of application acceleration could be even higher than normal web acceleration in the future.

- Acceleration in the pervasive networking environment

  With the explosive growth of the web, its application penetrates into more and more parts of people's life. Nowadays, Internet users are surfing the web from a wide different environments with different devices and preferences. For example, some people may use hand-held mobile device to surf the web, and some users may prefer to quickly browse through a compilation of news in simple textual format without spending long times to download the large images. This large variety of different requirements introduces new challenges to the acceleration of web content

delivery. It would be interesting to work along this direction as it can achieve the goal of acceleration in new environments.

- Peer-to-peer web system

  To exploit peer-to-peer techniques for web content delivery is a promising direction in terms of improving the latency, availability, and scalability etc. for web service. More works and good results may be expected from this direction in the future, especially for multimedia related content delivery.

Reference

[1] K. Thompson, G. Miller, and R. Wilder, Wide-area Internet traffic patterns and characteristics, Proceedings of Third International Conference on Web Caching, 1998.

[2] J. Hoe, Improving the Start-up Behavior of a Congestion Control Scheme for TCP, Proc. SIGCOMM '96, Aug. 1996.

[3] http://www.msnbc.com

[4] V. Paxson, Measurements and Analysis of End-to-End Internet Dynamics, PhD thesis, U.C. Berkeley, May 1996.

[5] The Need for Speed by Zona Research, July1999.

[6] http://www.w3.org/Protocols/HTTP/AsImplemented.html

[7] http://www.hitmill.com/Internet/web_history.asp

[8] Jia Wang, A survey of Web caching schemes for the Internet, ACM Computer Communication Review, 25(9):36-46, October 1999. The full version of the paper is published as Technical Report TR99-1747, Department of Computer Science, Cornell University, May 13, 1999.

[9] Bradley M. Duska, David Marwood, and Michael J. Feely, The Measured Access Characteristics of World-Wide-Web Client Proxy Caches, In Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97), December 1997.

[10] Duane Wessels, Web Caching, O'Reilly Publishing, 2001.

[11] Duane Wessels and K. Claffy, ICP and the Squid Web Cache, IEEE Journal on Selected Areas in Communication, 16(3):345-357, April 1998.

[12] Duchamp, D., Prefetching Hyperlinks, Proceedings of USENIX Symposium on Internet Technologies and Systems, October 1999.

[13] T.M. Kroeger, D.D.E. Long, and J.C. Mogul, Exploring the bounds of Web latency reduction from caching and prefetching, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems, Monterey, CA, Dec. 1997.

[14] Venkata N. Padmanabhan and Jeffrey C. Mogul, Using predictive prefetching to improve world wide web latency, Computer Communication Review, 26(3):22--36, July 1996.

[15] Content delivery market set to soar, http://www.nwfusion.com/news/2003/0106caching.html

[16] Web Applications Missing Link: Acceleration, http://www.atnewyork.com/news/article.php/1560491

[17] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox, Caching proxies: limitations and potentials, Proceedings of the 4th International WWW Conference, Boston, MA, Dec. 1995.

[18] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey C. Mogul, Rate of Change and other Metrics: a Live Study of the World Wide Web, In Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS'97), December 1997. An extended version is available as AT&T Labs - Research Technical Report 97.24.2.

[19] Themistoklis Palpanas, Balachander Krishnamurthy, Reducing Retrieval Latencies in the Web: the Past, the Present, and the Future, Technical Report CSRG-378, Department of Computer Science, University of Toronto, January 1999.

[20] Jun-Li Yuan and Chi-Hung Chi, Web Caching Performance: How Much Is Lost Unwarily?, LNCS Volume 2713/2003, pp. 23 - 33. Proceedings of the Second International Human.Society@Internet Conference, p.23-33, June 18 - 20, 2003, Seoul, Korea.

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, UC Irvine, Compaq/W3C, Compaq, W3C/MIT, Xerox, and Microsoft, June 1999

[22] Krishnamurthy, B., Rexford, J., Web Protocols and Practice, ISBN 0-201-71088-9, Addison-Wesley, 2001.

[23] Franks, J., Proposal for an HTTP MGET Method, http://ftp.ics.uci.edu/pub/ietf/http/hypermail/ 1994q4/0260.html, 1994.

[24] Venkata N. Padmanabhan and Jeffrey C. Mogul, Improving HTTP Latency, In Proceedings of the Second International World Wide Web Conference: Mosaic and the Web, pages 995--1005, Chicago, IL, October 1994. Also in Computer Networks and ISDN Systems,28:25-35, 1995.

[25] Craig E. Wills, Mikhail Mikhailov, and Hao Shang, N for the price of 1: Bundling web objects for more efficient content delivery, In Proceedings of the Tenth International World Wide Web Conference, Hong Kong, May 2001

[26] Gaurav Banga, Fred Douglis, and Michael Rabinovich, Optimistic Deltas for WWW Latency Reduction, In Proceedings of the USENIX Annual Technical Conference, USENIX Association, January 1997, pp. 289--304.

[27] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy, Potential Benefits of Delta-encoding and Data Compression for HTTP, In Proceedings of ACM SIGCOMM, pages 181--194, September 1997. An extended and corrected version appears as Research Report 97/4a, Digital Equipment Corporation Western Research Laboratory, December, 1997.

[28] Anubhav Savant and Torsten Suel, Server-Friendly Delta Compression for Efficient Web Access, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J. Watson Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[29] Mikhail Mikhailov and Craig E. Wills, Embedded objects in web pages, Technical Report WPI-CS-TR-00-05, Computer Science Department, Worcester Polytechnic Institute, March 2000.

[30] Pei Cao and Sandy Irani, Cost-Aware WWW Proxy Caching Algorithms, In Proceedings of the USENIX Symposium on Internet Technoloy and Systems, pages 193--206, December 1997.

[31] Binzhang Liu, Characterizing Web Response Time, Master Thesis, Computer Science, Virginia Polytechnic Institute and State University, April 22, 1998.

[32] Roland P. Wooster and Marc Abrams, Proxy caching that estimates page load delays, In Proceedings of the Sixth International World Wide Web Conference, pages 325-334, Santa Clara, CA, April 1997.

[33] Ahsan Habib, Marc Abrams, Analysis of Sources of Latency in Downloading Web Pages, In Proc. of WebNet Conference on the WWW and Internet (WebNet '00) San Antonio USA, Nov 2000

[34] Ruddle, A., Allison, C., Lindsay, P., Analysing the latency of WWW applications, Proceedings of the IEEE ICCCN, Phoenix, AZ, Oct., 2001.

[35] HyperText Markup Language (HTML) Home Page, http://www.w3.org/MarkUp/

[36] Hypertext Transfer Protocol -- HTTP/1.0, RFC 1945, http://www.faqs.org/rfcs/rfc1945.html

[37] Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, http://www.faqs.org/rfcs/rfc2616.html

[38] Transmission Control Protocol, RFC 793, http://www.faqs.org/rfcs/rfc793.html

[39] Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/1998/REC-xml-19980210

[40] Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml

[41] Robin Cover, WAP Wireless Markup Language Specification (WML), August 2001, http://www.oasis-open.org/cover/wap-wml.html

[42] Wireless Markup Language 2.0, http://www1.wapforum.org/tech/documents/WAP-238-WML-20010626-p.pdf

[43] Nottingham, M., Liu, X., Edge Architecture Specification, W3C Note 04, August 2001, http://www.w3.org/TR/edge-arch

[44] ESI¡§CAccelerating E-Business Applications, http://www.edge-delivery.org

[45] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., Web Service Definition Language (WSDL), W3C Note 15, March 2001, http://www.w3.org/TR/wsdl

[46] D. Wessels, K. Claffy, Internet Cache Protocol(ICP), version 2, Network Working Group, RFC: 2186, 1997.

[47] D. Wessels, K. Claffy, Application of Internet Cache Protocol(ICP), version 2, Network Working Group, RFC: 2187, 1997.

[48] P. Vixie, D. Wessels, Hyper Text Caching Protocol(HTCP/0.0), Network Working Group, RFC: 2756, Jan. 2000.

[49] Internet Content Adaptation Protocol (I-CAP), http://www.i-cap.org

[50] ICAP press releases, http://www.i-cap.org/icap/press.cfm

[51] Tomlinson, G., Orman, H., Condry, M., Kempf, J., Farber, D., Extensible Proxy Services Framework, IETF-OPES Internet drafts, July 2000, http://www.ietf-opes.org/documents/draft-tomlinson-epsfw-00.txt

[52] Yang, L., Hofmann, M., OPES Architecture fro Rule Processing and Service Execution, IETF-OPES Internet drafts, 2000, http://www.ietf-opes.org/documents/draft-yang-opes-rule-processing-service-execution-00.txt

[53] Tomlinson, G., Chen, R., Hofmann, M., A Model for Open Pluggable Edge Services, IETF 51, London , July 2001

[54] McHenry, S., Condry, M., Tomlinson, G., Orman, H., Hoffman, M., Open Pluggable Edge Services Use Cases and Deployment Scenarios, IETF 51, London, July 2001

[55] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D., Simple Object Access Protocol (SOAP) 1.1, W3C Note 08, May 2000, http://www.w3.org/TR/SOAP

[56] Mark Nottingham, SOAP Optimisation Modules: Response Caching, W3C Archives, 2001, http://lists.w3.org/Archives/Public/www-ws/2001Aug/att-0000/01-ResponseCache.html

[57] Web Intermediaries (webi) Charter, IETF Internet drafts, http://www.ietf.org/html.charters/webi-charter.html

[58] McManus, P., Nottingham, M., Requirements for Intermediary Discovery and Description, IETF Internet drafts, February 2001, http://www.ietf.org/Internet-drafts/draft-ietf-webi-idd-reqs-00.txt

[59] Hamilton, M., Cooper, I., Li, D., Requirements for a Resource Update Protocol, IETF Internet drafts, July 2001, http://www.ietf.org/Internet-drafts/draft-ietf-webi-rup-reqs-01.txt

[60] Web Replication and Caching (WREC), IETF Internet drafts, http://www.ietf.org/html.characters/wrec-charter.html

[61] Cooper, I., Melve, I., Tomlinson, G., Internet Web Replication and Caching Taxonomy, IETF Internet drafts, January 2001, http://www.ieft.org/rfc/rfc3040.txt

[62] Middlebox Communication (midcom) Charter, IETF Internet drafts, http://www.ietf.org/html.charters/midcom-charter.html

[63] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., Rayhan, A., Middlebox Communication Architecture and framework, IETF Internet drafts, July 2001, http://www.ietf.org/Internet-drafts/draft-ietf-midcom-framework-03.txt

[64] Reliable Server Pooling (rserpool) Charter, IETF Internet drafts, http://www.ietf.org/html.charters/rserpool-charter.html

[65] Tuexen, M., Xie, Q., Stewart, R., Shore, M., Ong, L., Loughney, J., Stillman, M., Requirements for Reliable Server Pooling, IETF Internet drafts, May 2001, http://www.ietf.org/rfc/rfc3237.txt

[66] Tuexen, M., Xie, Q., Stewart, R., Shore, M., Ong, L., Loughney, J., Stillman, M., Architecture for Reliable Server Pooling, IETF Internet drafts, April 2002, http://www.ietf.org/Internet-drafts/draft-ietf-rserpool-arch-02.txt

[67] Loughney, J., Stillman, M., Tuexen, M., Xie, Q., Stewart, R., Ong, L., Comparison of Protocols for Reliable Server Pooling, IETF Internet drafts, March 2002, http://www.ietf.org/Internet-drafts/draft-ietf-rserpool-comp-03.txt

[68] Stewart, S., Xie, Q., Aggregate Server Access Protocol (ASAP), IETF Internet drafts, March 2002, http://www.ietf.org/Internet-drafts/draft-ietf-rserpool-asap-03.txt

[69] Q. Xie, R. Stewart, Endpoint Name Resolution Protocol, IETF Internet drafts, May 2002, http://www.ietf.org/proceedings/01dec/slides/rserpool-2/

[70] Microsoft Corporation, http://www.microsoft.com

[71] Netscape, http://www.netscape.com

[72] Apache Group, Apache HTTP server documentation, http://www.apache.org/

[73] Squid Web Proxy Cache, http://www.squid-cache.org/

[74] Duane Wessels, Squid: The Definitive Guide, January 2004, O'Reilly and Associates, ISBN 0-596-00162-2.

[75] Httpd, http://www.w3.org/Daemon/User/Proxies/Proxies.html

[76] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox, Removal Policies in Network Caches for World-Wide Web Documents, In Proceedings of ACM SIGCOMM, pages 293-305, Stanford, CA, 1996. Revised March 1997.

[77] Pei Cao and Sandy Irani, GreedyDual-Size: A Cost-Aware WWW Proxy Caching Algorithm, 2nd Web Caching Workshop, Boulder, Colorado, June 1997.

[78] S. G. Dykes, Cooperative Web Caching: A Viability Study and Design Analysis, Ph.D. Dissertation, University of Texas at San Antonio, Aug. 2000.

[79] S. G. Dykes, Cooperative Web Caching: A Viability Study and Design Analysis, Talk slides from the dissertation defense, University of Texas at San Antonio, Aug. 2000.

[80] S. G. Dykes, K. A. Robbins, and C. L. Jeffery, Uncacheable documents and cold starts in Web proxy cache simulations: How two wrongs appear right, Technical Report CS-2001-01, University of Texas at San Antonio, Division of Computer Science, San Antonio, TX 78249-0664, Jan. 2001.

[81] Mark Nottingham, Optimizing Object Freshness Controls in Web Caches, In 4th International Web Caching Workshop (WCW'99), San Diego, CA, March 31 - April 2 1999.

[82] E. Cohen and H. Kaplan, Refreshment policies for Web content caches, In Proceedings of the IEEE INFOCOM'01 Conference. 2001.

[83] Yin J., Alvisi L., et al., Volume Leases for Consistency in Large-Scale Systems, IEEE Transactions on Knowledge Data Engineering, July 1999, Vol. 11, No. 4, pp. 563-576.

[84] Carlos Maltzahn, Kathy J. Richardson, Dirk Grunwald and James H. Martin, On Bandwidth Smoothing, In Proceedings of the 4th International Web Caching Workshop, San Diego, CA, Mar. 1999.

[85] Carlos Maltzahn, Kathy J. Richardson, Dirk Grunwald, and James Martin, A Feasibility Study of Bandwidth Smoothing on the World-Wide Web Using Machine Learning, Technical report #CU-CS-879-99, Dept. of Computer Science, University of Colorado at Boulder, January, 1999.

[86] T. Palpanas and A. Mendelzon, Web Prefetching Using Partial Match Prediction, Proceedings 4th Web Caching Workshop, San Diego, CA, March 1999.

[87] Ramesh R. Sarukkai, Link prediction and path analysis using markov chains, In Proceedings of 9th International World Wide Web Conference, 2000.

[88] Mukund Deshpande and George Karypis, Selective Markov Models for Predicting Web-Page Accesses, 1st SIAM Data Mining Conference, 2001

[89] Evangelos P. Markatos and Catherine E. Chronaki, A top-10 approach for prefetching the web, In Proceedings of the Eighth Annual Conference of the Internet Society (INET'98), Geneva, Switzerland, July 1998. Also available as ICS-FORTH Technical Report 173.

[90] A. J. Smith, Cache memories, ACM Computing Surveys, vol. 14, pp. 473-530, Sept. 1982.

[91] Fredrick J. Hill and Gerald R. Peterson, Digital Systems: Hardware Organization and Design, John Wiley & Sons, New York, 1987. Third Edition.

[92] M. Morris Mano, Computer System Architecture, Prentice-Hall, Englewood Cliffs, NJ, 1982. Second Edition.

[93] Miles J. Murdocca and Vincent P. Heuring, Principles of Computer Architecture, Addison Wesley Longman, To appear, 1999.

[94] Edith Cohen and Haim Kaplan, Caching documents with varying sizes and fetching costs: an LP-based approach, Algorithmica , 32(3):459-466, 2002.

[95] G.Z. Chrysos, J.S. Emer, Memory dependence prediction using store sets, Proceedings of the 25th Annual International Symposium on Computer Architecture, ACM, New York, 1998, pp. 142?¡ìC153.

[96] Chi-Hung Chi and Jun-Li Yuan, Load-Balancing Branch Target Cache and Prefetch Buffer, Proceedings of the 1999 IEEE International Conference on Computer Design (ICCD 1999), p. 436-441, Austin, Texas, October 10-13, 1999.

[97] Chi-Hung Chi and Jun-Li Yuan, Sequential Unification and Aggressive Lookahead Mechanisms for Data Memory Accesses, Proceedings of the 5th International Conference on Parallel Computing Technologies (PaCT-99), p.28-41, St. Petersburg, Russia, September 6-10, 1999.

[98] Chi-Hung Chi and Jun-Li Yuan, Design Considerations of High Performance Data Cache with Prefetching, Euro-Par 1999: 1243-1250, September 1999, LNCS 1685.

[99] Chi-Hung Chi , Jun-Li Yuan and Chin-Ming Cheung, Cyclic dependence based data reference prediction, Proceedings of the 13th International Conference on Supercomputing, p.127-134, June 20-25, 1999, Rhodes, Greece.

[100] Chi-Hung Chi and Jun-Li Yuan, Runtime Association of Software Prefetch Control to Memory Access Instructions, Euro-Par 2002: 486-489.

[101] Chi-Hung Chi and Jun-Li Yuan, Load-balancing data prefetching techniques, Invited paper in Journal of Future Generation Computer Systems, April 2001, Volume 17 Issue 6 p.733-44.

[102] Pei Cao, Characterization of Web Proxy Traffic and Wisconsin Proxy Benchmark 2.0, In World Wide Web Consortium Workshop on Web Characterization, Cambridge, MA, November 1998. Position paper.

[103] Lee Breslau, Pei Cao, Li Fan, Graham Phillips and Scott Shenker, Web Caching and Zipf-like Distributions: Evidence and Implications, In Proceedings of the IEEE Infocom '99 Conference, New York, NY, March 1999. 148.

[104] S. G. Dykes and K. A. Robbins, Correcting the application of Zipf's Law to Web proxy caching, SIGCOMM 2000 poster presentation

[105] Ghaleb Abdulla, Edward A. Fox, Marc Abrams, and Stephen Williams, WWW Proxy Traffic Characterization with Application to Caching, Technical Report TR-97-03, Computer Science Dept., Virginia Tech, Mar. 1997.

[106] Ghaleb Abdulla, A. H. Nayfeh, and Edward A. Fox, A Realistic Model of Request Arrival Rate to Caching Proxies, Submitted for publication, 1997. http://vtopus.cs.vt.edu/~chitra/docs/abdulla-nayfeh-fox/paper.pdf

[107] Ghaleb Abdulla, Analysis and Modeling of World Wide Web Traffic, PhD Dissertation, Virginia Polytechnic Institute and State University, 1998.

[108] Arthur Goldberg, Ilya Pevzner, and Robert Buff, Caching Characteristics of Internet and Intranet Web Proxy Traces, Published in the Computer Measurement Group Conference, CMG98, December 1998

[109] Terence Kelly, Thin-client Web access patterns: Measurements from a cache-busting proxy, In Web Caching and Content Delivery: Proceedings of the Sixth International Web Content Caching and Distribution Workshop (WCW'01), Boston, MA, June 2001.

[110] Craig E. Wills and Mikhail Mikhailov, Towards a better understanding of web resources and server responses for improved caching, In Proceedings of the 8th International World Wide Web Conference,Toronto, Canada, pages 153--165, May 1999.

[111] Xiaohui Zhang, Cachability of Web Objects, Technical Report 2000-019, Computer Science Department, Boston University, August 8, 2000.

[112] Craig E. Wills and Mikhail Mikhailov, Examining the cacheability of user-requested web resources, In Proceedings of the 4th International Web Caching Workshop, pages 78-87, San Diego, CA, March/April 1999.

[113] Jeffrey C. Mogul, Errors in timestamp-based HTTP header values, Technical Research Report 99/3, Compaq Western Research Lab, December 1999.

[114] Ludmila Cherkasova, Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy, Hewlett-Packard Company Report, Computer Systems Laboratory, HPL-98-69 (R.1), November, 1998

[115] S. Jin and A. Bestavros, Popularity-aware greedy-dual-size web proxy caching algorithms, In Proceedings of ICDCS'2000: The IEEE International Conference on Distributed Computing Systems, Taiwan, May 2000.

[116] Shudong Jin and Azer Bestavros, GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams, International Journal on Computer Communications, 24(2):174-183, February 2001.

[117] Luigi Rizzo and Lorenzo Vicisano, Replacement policies for a proxy cache, IEEE/ACM Transactions on Networking, 8(2):158--170, 2000. (The same as RV98)

[118] Luigi Rizzo and Lorenzo Vicisano, Replacement policies for a proxy cache, Research Note RN/98/13, Department of Computer Science, University College London, 1998. (The same as RV00)

[119] Peter Scheuermann, Junho Shim, and Radek Vingralek, A case for delay-conscious caching of Web documents, In Proceedings of the Sixth International World Wide Web Conference, Santa Clara, CA, April 1997.

[120] J. Dilley, M. Arlitt and S. Perret, Enhancement and Validation of the Squid's Cache Replacement Policy, Proceeding of the Fourth Web Caching Workshop, San Diego, March 1999. Also available as HP Labs Technical Reports, HPL-1999-69, 990527, at http://www.hpl.hp.com/techreports/1999/HPL-1999-69.html

[121] Chengjie Liu and Pei Cao, Maintaining strong cache consistency in the world-wide web, In Proceedings of ICDCS'97, pages 12--21, May 1997, URL: http://www.cs.wisc.edu/~cao/papers/icache.html.

[122] James Gwertzman and Margo Seltzer, World-Wide Web Cache Consistency, In Proceedings of the USENIX Technical Conference, San Diego, CA, January 1996.

[123] V. Cate, Alex--- A global filesystem, In Proceedings of the USENIX File System Workshop, pages 1--12, Ann Arbor, MI, May 1992.

[124] J. Gwetzman and M. Seltzer, The case for geographical pushing-caching, HotOS Conference, 1994.

[125] Balachander Krishnamurthy and Craig E. Wills, Study of piggyback cache validation for proxy caches in the world wide web, In Symposium on Internet Technologies and Systems. USENIX Association, December 1997.

[126] Balachander Krishnamurthy and Craig E. Wills, Piggyback server invalidation for proxy cache coherency, In Proceedings of the Seventh International World Wide Web Conference, pages 185-193, Brisbane, Australia, April 1998.

[127] Mikhail Mikhailov and Craig E. Wills, Evaluating a new approach to strong web cache consistency with snapshots of collected content, In Proceedings of the Twelfth International World Wide Web Conference, Budapest, Hungary, May 2003.

[128] Ronald Dodge and Daniel A. Menasce, Prefetching Inlines To Improve Web Server Latency, In the Proceedings of the 1998 Computer Measurement Group Conference, Anaheim, CA, Dec. 6-11, 1998.

[129] Ken-ichi Chinen and Suguru Yamaguchi, An interactive prefetching proxy server for improvement of WWW latency, In Proceedings of the Seventh Annual Conference of the Internet Society (INET'97), Kuala Lumpur, June 1997.

[130] Azer Bestavros, Using Speculation to Reduce Server Load and Service Time on the WWW, In Proceedings of CIKM'95: The Fourth ACM International Conference on Information and Knowledge Management, Baltimore, MD, November 1995. Also available as Technical Report TR-95-006, Computer Science Department, Boston University.

[131] Zhimei Jiang and Leonard Keinrock, Prefetching Links on the WWW, In ICC'97, pages 483--489, Montreal, Canada, June 1997.

[132] Tong Sau Loon and Vaduvur Bharghavan, Alleviating the latency and bandwidth problems in www browsing, In Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, Monterey, CA, December 1997.

[133] Craig E. Wills and Joel Sommers, Prefetching on the web through merger of client and server profiles, June 1997.

[134] Stuart Schechter, Murali Krishnan, and Michael D. Smith, Using Path Profiles to predict http requests, In 7th International World Wide Web Conference, pages 457--467, Brisbane, Qld., Australia, April 1998.

[135] E. Cohen, B. Krishnamurthy, and J. Rexford, Efficient algorithms for predicting requests to web servers, In Proceedings of IEEE INFOCOM, March 1999.

[136] B. D. Davison, Topical Locality in the Web: Experiments and Observations, Technical Report DCS-TR-414, Department of Computer Science, Rutgers University.

[137] Sajid Hussain, Intelligent Prefetching, Graduate Students Conference, GRADCON'99, Winnipeg, MB, Canada; October 1, 1999.

[138] Suyoung Yoon, Eunsook Jin, Jungmin Seo and Ju-Won Song, Prefetching Brand-new Documents for Improving the Web Performance, In Proceedings of

the 9th Annual Conference of the Internet Society, INET'99, San Jose, US, June 1999.

[139] A. Eden, B. Joh, T. Mudge, Web Latency Reduction via Client-Side Prefetching, In Proceedings of 2000 IEEE Int. Symp. on Perfor-mance Analysis of Systems & Software (ISPASS-2000), Austin, TX, pp. 193-200

[140] Zhong Su, Qiang Yang, Ye Lu and Hong Jiang Zhang, WhatNext: A Prediction System for Web Requests using N-gram Sequence Models, In First International Conference on Web Information Systems and Engineering Conference. Hong Kong, June 2000.

[141] Michael Zhen Zhang and Qiang Yang, Model-based Predictive Prefetching, In Proceedings of the 2nd International Workshop on Management of Information on the Web -- Web Data and Text Mining (MIW'01). September 2001. Munich, Germany; 3-7 September, 2001.

[142] B. D. Davison, Predicting Web Actions from HTML Content, In Proceedings of the The Thirteenth ACM Conference on Hypertext and Hypermedia (HT'02), College Park, MD, June 11-15, pages 159-168.

[143] Darin Fisher, Gagan Saksena, Link Prefetching in Mozilla: A Server-Driven Approach, SYNOPSIS, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J. Watson Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[144] Mark Crovella and Paul Barford, The Network Effects of Prefetching, In Proceedings of IEEE Infocom '98, San Francisco, CA, 1998. More detailed version available as Boston University Computer Science Department Technical Report, TR-97-002, February 1997.

[145] B. D. Davison, Assertion: Prefetching With GET Is Not Good, In A. Bestavros and M. Rabinovich (eds), Web Caching and Content Delivery: Proceedings of the Sixth International Web Content Caching and Content Distribution Workshop (WCW'01), Boston, June 20-22, 2001, pages 203-215, Elsevier.

[146] Arun Venkataramani, Praveen Yalagandula, Ravindranath Kokku, Sadia Sharif, and Mike Dahlin, Potential costs and benefits of long-term prefetching for content-distribution, Computer Communications Journal, 25(4):367--375, 2002.

[147] Li Fan, Quinn Jacobson, and Pei Cao, Potential and limits of web prefetching between low-bandwidth clients and proxies, In Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, 1999.

[148] Yingyin Jiang, Min-You Wu, Wei Shu, JPEG2000 offers new opportunities to enrich image content and applications flexibility, 7th International Workshop on Web Content Caching and Distribution (WCW) Boulder, Colorado, August 14-16, 2002.

[149] Ajay B Pandey, Ranga R Vatsavai, Xiaobin Ma, Jaideep Srivastava, Shashi Shekhar, A Comparative Study of Web Prefetching Algorithms, Submitted to the special issue of the IEEE Journal on Selected Areas in Communications on Internet Proxy Services (May 1, 2001).

[150] Radhika Malpani, Jacob Lorch and David Berger, Making World Wide Web Caching Servers Cooperate, In Proceedings of the 4th International World Wide Web Conference, Boston, Dec 1995.

[151] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrel, A hierarchical Internet object cache, Usenix'96, January 1996.

[152] E. Cohen, E. Halperin, and H. Kaplan, Performance aspects of distributed caches using TTL-based consistency, In Proceedings of the ICALP'01 conference, Springer-Verlag, LNCS. 2001.

[153] E. Cohen and H. Kaplan, The age penalty and its effect on cache performance, In Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS). 2001.

[154] E. Cohen and H. Kaplan, Aging through cascaded caches: performance issues in the distribution of Web content, In Proceedings of the ACM SIGCOMM'01 Conference . 2001.

[155] Sandra G. Dykes, Clinton L. Jeffery and Samir Das, Taxonomy and Design Analysis for Distributed Web Caching, In the Proceedings of the Hawaii International Conference on System Sciences, January 5-8, 1999, Maui, Hawaii.

[156] S. G. Dykes and K. A. Robbins, A Viability analysis of coopertive proxy caching, IEEE Infocom 2001, Vol. 3, Apr. 2001, pp.1205-1214

[157] S. G. Dykes and K. A. Robbins, Limitations and benefits of cooperative proxy caching, IEEE Journal on Selected Areas in Communications (J-SAC) to appear (2001?)

[158] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder, Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol, In Proceedings of ACM SIGCOMM, September 1998.

[159] Li Fan, Pei Cao, Wei Lin and Quinn Jocobson, Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance, In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99), Atlanta, GA, May 1999.

[160] Michal Kurcewicz, Wojtek Sylwestrzak, and Adam Wierzbicki, A filtering algorithm for proxy caches, In Third International WWW Caching Workshop, Manchester, England, June 1998.

[161] Hyokyung Bahn, Hyunsook Lee, Sam H. Noh, Sang Lyul Min, and Kern Koh School, Replica-aware caching for Web proxies, Computer Communications, 25(3):183--188, Feb. 2002.

[162] Terence Kelly and Jeff Mogul, Aliasing on the World Wide Web: Prevalence and Performance Implications, In Proceedings of The Eleventh International World Wide Web Conference, Honolulu, Hawaii, 7-11 May 2002.

[163] Jeffrey C. Mogul, A trace-based analysis of duplicate suppression in HTTP, Research Report 99/2, COMPAQ, Western Research Laboratory, Nov. 1999.

[164] Jeffrey C. Mogul, Squeezing More Bits Out of HTTP Caches, IEEE Network 14(3):6-14, May/June, 2000.

[165] Hua Chen, Marc Abrams, Tommy Johnson, Anup Mathur, Ibraz Anwar, and John Stevenson, Wormhole Caching with HTTP PUSH Method for a SatelliteBased Web Content Multicast and Replication Syste, In Proceedings of 4th International WWW Caching Workshop, San Diego, California, March 31 - April 2 1999. http://www.ircache.net/Cache/Workshop99/Papers/chen-html/

[166] T. Loukopoulos, P. Kalnis, I. Ahmad and D. Papadias, Active Caching of On Line Analytical Processing Queries in WWW Proxies, In Proc. of the Int.

Conference on Parallel Processing (ICPP), Valencia, Spain, 419-426, 2001. (Best Paper Award)

[167] Evangelos P. Markatos, On Caching Search Engine Query Results, Technical Report 241, Institute of Computer Science, Foundation for Research & Technology, Greece, 1999.

[168] Evangelos P. Markatos, On Caching Search Engine Query Results, In Proceedings of the 5th International Web Caching and Content Delivery Workshop, May 2000

[169] Mor Naaman, Hector Garcia-Molina, Andreas Paepcke, Evaluation of ESI and Class-Based Delta Encoding, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J. Watson Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[170] P. Cao, J. Zhang, and K. Beach, Active cache: caching dynamic contents on the Web, Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), pp. 373-388.

[171] Songqing Chen and Xiaodong Zhang, Detective Browsers: A Software Technique to Improve Web Access Performance and Security, 7th International Workshop on Web Content Caching and Distribution (WCW), Boulder, Colorado, August 14-16, 2002

[172] Chi Hung Chi and HongGuang Wang, A Generalized Model for Characterizing Content Modification Dynamics of Web Objects, SYNOPSIS, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J. Watson Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[173] Mikhail Mikhailov and Craig E. Wills, Change and Relationship-Driven Content Caching, Distribution and Assembly, Technical Report (WPI-CS-TR-01-03), WORCESTER POLYTECHNIC INSTITUTE, Computer Science Department, March 2001.

[174] Chun Yuan, Zhigang Hua and Zheng Zhang, Proxy+: Simple Proxy Augmentation for Dynamic Content Processing, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J.

Watson Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[175] Huican Zhu and Tao Yang, Class-Based Cache Management for Dynamic Web Content, In Proceedings of the IEEE Infocom 2001 Conference, Anchorage, Alaska USA, April 2001.

[176] Arthur Goldberg, Robert Buff, and Andrew Schmitt, A Comparison of HTTP and HTTPS Performance, Published in the Computer Measurement Group, CMG98, December 1998.

[177] Arthur Goldberg, Robert Buff, and Andrew Schmitt, Secure Web Server Performance Dramatically Improved By Caching SSL Session Keys, Published in the Workshop on Internet Server Performance, held in conjunction with SIGMETRICS'98, June 23, 1998

[178] Jussi Kangasharju, James W. Roberts, and Keith W. Ross, Object Replication Strategies in Content Distribution Networks, Computer Communications, Volume 25, Number 4, March 2002. pp. 367-383, 2002.

[179] Sven Buchholz and Thomas Buchholz, Replica Placement in Adaptive Content Distribution Networks, In ACM Symposium on Applied Computing (SAC'04), Nicosia, Cyprus, March 2004.

[180] Zongming Fei, A Novel Approach to Managing Consistency in Content Distribution Networks, In Proceedings of Web Caching and Content Distribution Workshop (WCW'01), Boston, MA, June 2001.

[181] Kirk Johnson, John Carr, Mark Day, and Frans Kaashoek, The Measured Performance of Content Distribution Networks, In Fifth International Web Caching and Content Delivery Workshop, Lisbon, Portugal, May 2000.

[182] Jussi Kangasharju, Keith W. Ross, and Jim W. Roberts, Performance Evaluation of Redirection Schemes in Content Distribution Networks, In Fifth International Web Caching and Content Delivery Workshop, Lisbon, Portugal, May 2000.

[183] Balachander Krishnamurthy, Craig Wills and Yin Zhang, On the Use and Performance of Content Distribution Networks, In Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW'2001), November 2001.

[184] Jacobus Van der Merwe, Paul Gausman, Chuck Cranor, Rustam Akhmarov, Design, Implementation and Operation of a large Enterprise Content Distribution Network, SYNOPSIS, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J. Watson Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[185] Sampath Rangarajan, Pablo Rodriguez, Sarit Mukherjee, User Specific Request Redirection in a Content Delivery Network, SYNOPSIS, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J. Watson Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[186] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, Web proxy caching: the devil is in the details, ACM Performance Evaluation Review, 26(3): pp. 11-15, December 1998.

[187] Virglio Almeida, Daniel Menasc¡§|, Rudolf Riedi, Fl¡§¡évia Peligrinelli, Rodrigo Fonseca, Wagner Meira, Jr., Analyzing Web Robots and their Impact on Caching, Proc. Sixth Workshop on Web Caching and Content Distribution, June, 2001, pp. 299--310.

[188] Balachander Krishnamurthy and Craig E. Wills, Analyzing factors that influence end-to-end web performance, Worcester Polytechnic Insitute, Computer Science, Technical Report, WPI-CS-TR-99-35, Nov. 1999.

[189] Balachander Krishnamurthy and Craig E. Wills, Analyzing factors that influence end-to-end web performance, In Proceedings of the Ninth International World Wide Web Conference, Amsterdam, Netherlands, May 2000.

[190] Binzhang Liu and Edward A. Fox, Web Traffic Latency: Characteristics and Implications, Journal of Universal Computer Science, vol. 4, no. 9 (1998), 763-778.

[191] Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Wium Lie, and Chris Lilley, Network Performance Effects of HTTP/1.1, CSS1, and PNG, In Proc. SIGCOMM'97. Cannes, France, September, 1997.

[192] E. Cohen, H. Kaplan, and U. Zwick, Connection Caching, In Proceedings of the 31 st Annual ACM Symposium on Theory of Computing, Atlanta, Georgia, May 1999, pp. 612-621.

[193] Craig E. Wills and Hao Shang, The contribution of DNS lookup costs to web object retrieval, Technical Report WPI-CS-TR-00-12, Computer Science Department, Worcester Polytechnic Institute, July 2000.

[194] Girish Chandranmenon, Reducing web latencies using precomputed hints, Tech. Rep. PhD Thesis. Technical report WUCS-99-18, Dept of Computer Science, Washington University in St. Louis, August 1999.

[195] 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, March 2000.

[196] 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 (SAINT-2001), IEEE, San Diego, January 2001.

[197] Jeffrey C. Mogul, The Case for Persistent-Connection HTTP, In Proceedings of the ACM SIGCOMM '95 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication, pages 299-313, 1995.

[198] Susanne Albers, Generalized Connection Caching, SPAA 2000, Bar Harbor, Maine USA, Copyright ACM 2000 1-58113-185-2/00/07

[199] E. Cohen, H. Kaplan, and J. D. Oldham, Managing TCP Connections under Persistent HTTP, Computer Networks. 31:1709--1723, 1999.

[200] E. Cohen, H. Kaplan, and U. Zwick, Connection caching under various models of communication, In Proc. 12th Annual ACM Symposium on Parallel Algorithms and Architectures. ACM, 2000.

[201] Craig E. Wills, Gregory Trott, and Mikhail Mikhailov, Using bundles for web content delivery, Computer Networks, 42(6):797-817, August 2003.

[202] Chi Hung Chi, HongGuang Wang and William Ku, Proxy-Cache Aware Object Bundling for Web Access Acceleration, In Proceedings of the 8th International Workshop on Web Content Caching and Distribution, IBM T.J. Watson

Research Center, Hawthorne, NY USA, 29 September - 1 October 2003. http://2003.iwcw.org/

[203] Mihut D. Ionescu, xProxy: A transparent caching and delta transfer system for web objects, May 2000. UC Berkeley class project: CS262B/CS268. http://www.cs.pdx.edu/~delco/xproxy.ps.gz

[204] Zan Ouyang, Nasir Memon, Torsten Suel, and Dimitre Trendafilov, Cluster-based delta compression of a collection of files, In Third Int. Conf. on Web Information Systems Engineering, December 2002.

[205] Jun-Li Yuan and Chi-Hung Chi, Unveiling the Performance Impact of Lossless Compression to Web Page Content Delivery, LNCS Volume 3293/2004, pp. 249 - 260. Proceedings of the Ninth International Workshop on Web Content Caching and Distribution (WCW 2004), Beijing, China, 18-20 October 2004.

[206] HTTP Compression Speeds up the Web, http://www.webreference.com/Internet/software/servers/http/compression/

[207] Using HTTP Compression On Your IIS 5.0 Web Site, http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/iis/maintain/featusability/httpcomp.asp

[208] Apache Gzip Module from Mozilla, http://www.mozilla.org/projects/apache/gzip/

[209] DEFLATE Compressed Data Format Specification, RFC 1951, http://www.faqs.org/rfcs/rfc1951.html

[210] gzip home page, http://www.gzip.org

[211] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, May 1977.

[212] Terry Welch, A Technique for High-Performance Data Compression, Computer, June 1984.

[213] GZIP file format specification, RFC 1952, http://www.faqs.org/rfcs/rfc1952.html

[214] zlib home page, http://www.gzip.org/zlib/

[215] Packeteer?¡¥s PacketShaper Xpress, http://www.packeteer.com/prod-sol/products/xpress.cfm

[216] The Effect of HTML Compression on a LAN and a PPP Modem Line, http://www.R27/Protocols/HTTP/Performance/Compression/LAN.html , http://www.R27/Protocols/HTTP/Performance/Compression/PPP.html

[217] Ronny Krashinsky, Efficient web browsing for mobile clients using HTTP compression, Distributed Operating Systems term project, Massachusetts Institute of Technology, December 2000.

[218] Surendar Chandra and Carla Schlatter Ellis, JPEG compression metric as a quality-aware image transcoding, In Proc. USENIX 2nd Symposium on Internet Technology and Systems, pages 81-92, Boulder, CO, Oct. 1999.

[219] Armando Fox, Eric A Brewer, Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation, Proceedings of Fifth International World Wide Web Conference, 1996.

[220] H. Bharadvaj, A. Joshi and S. Auephanwiriyakul, An active transcoding proxy to support mobileWeb access, Proceedings of 17th IEEE Symposiumon Reliable Distributed Systems, October 1998.

[221] S. Chandra, C. S. Ellis and A. Vahdat, Differentiated multimedia Web services usingquality aware transcoding, Proceedings of INFOCOM 2000 - Nineteenth Annual JointConference of the IEEE Computer AndCommunications Societies, 2000.

[222] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives, Proceedings of ASPLOS-VII, 1996.

[223] A. Joshi, On proxy agents, mobility, and Web access, In ACM/Baltzer Journal of MobileNetworks and Nomadic Applications(MONET), December, 2000.

[224] Free Web Site Acceleration, http://siliconvalley.Internet.com/news/article.php/484971

[225] Platform for Internet Content Selection (PICS), http://www.w3.org/PICS/

[226] http://monitor.optiview.com/POV/task,ov4optimizationworks/parse.html

[227] http://www.pipeboost.com/home.html

[228] Content Selection for Device Independence (DISelect) 1.0, W3C Working Draft 11 June 2004, http://www.w3.org/TR/2004/WD-cselection-20040611/, http://www.w3.org/TR/cselection/

[229] Rodriguez, P., Kirpal, A., Biersack, E.W., Parallel-Access for Mirror Sites in the Internet, Proceedings of IEEE INFOCOM 2000 Conference, March 2000.

[230] Miu, A., Shih, E., Performance Analysis of a Dynamic Parallel Downloading Scheme from Mirror Sites Throughout the Internet, Term Paper, LCS MIT, December 1999.

[231] Rodriguez, P., Biersack, E.W., Dynamic Parallel-Access to Replicated Content in the Internet, IEEE/ACM Transactions on Networking, August 2002.

[232] B. D. Davison and V. Liberatore, Pushing Politely: Improving Web Responsiveness One Packet at a Time, In Performance Evaluation Review, Volume 28, Number 2, September 2000, pages 43-49. Presented at the Performance and Architecture of Web Servers (PAWS) Workshop, held in conjunction with ACM SIGMETRICS 2000: International Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA, June 17-18.

[233] C. Christopoulos, A. Skodras, and T. Ebrahimi, The JPEG2000 still image coding system: an overview, IEEE Transactions on Consumer Electronics, Vol 46, No. 4, pp. 1103-1127, November 2000.

[234] D. S. Cruz and T. Ebrahimi, An analytical study of JPEG2000 Functionalities, Proceedings of IEEE International Conference on Image Processing. September 2000.

[235] The JPEG group?¡¥s official homepage, http://www.jpeg.org

[236] JPEG 2000 White Paper prepared by Digital Imaging Group, JPEG2000 offers new opportunities to enrich image content and applications flexibility, http://www.ecs.soton.ac.uk/~km/docs/jpeg2000.doc

[237] Gettys, J., Nielsen, H.F., The WebMUX protocol, Internet Draft, August 1998. http://www.w3.org/Protocols/MUX/WDmux- 980722.html

[238] The Hypertext Streaming Transport Protocol, http://netlab.cis.temple.edu/bxxp/hstp.html

[239] J. Franks, P. Hallan-Baker et el, An Extension to HTTP: Digest Access Authentication, Network Working Group, RFC: 2069, Jan. 1997.

[240] Wenting Tang, Ludmila Cherkasova et el, Modular TCP Handoff Design in STREAMS-Based TCP/IP Implementation, IEEE 2001 International Conference on Networking (ICN'01), July 9-13, 2001.

[241] Stream Control Transmission Protocol (SCTP), http://www.sctp.org/

[242] The Blocks Extensible Exchange Protocol Core, http://xml.resource.org/public/rfc/html/rfc3080.html, http://www.beepcore.org/

[243] R. J. Bayardo Jr., A. Somani, D. Gruhl, and R. Agrawal, YouServ: A Web Hosting and Content Sharing Tool for the Masses, In Proc. of the 11th Int'l World Wide Web Conference (WWW-2002), 2002. http://www.almaden.ibm.com/cs/people/bayardo/userv/userv.html

[244] R. J. Bayardo Jr., A. Costea, and R. Agrawal, Peer-to-Peer Sharing of Web Applications, IBM Research Report RJ 10268, Nov. 2002. Poster version appears in Proc. of the 12th Int'l World Wide Web Conference (WWW-2003), Budapest, Hungary, May 2003. http://www.almaden.ibm.com/cs/people/bayardo/userv/plugins/plugin.html

[245] BadBlue P2P web server adds Gnutella support, http://www.infoanarchy.org/?op=displaystory&sid=2002/2/17/141113/123

[246] Blue Coat Systems, http://www.bluecoat.com

[247] http://www.netapp.com/products/netcache/netcache_family.html

[248] Cisco Systems, Inc., Cisco cache engine, Available at http://www.cisco.com/warp/public/751/cache/, 1998.

[249] G. Tomlinson, D. Major, and R. Lee, High-capacity Internet middleware: Internet caching system architectural overview, Second Workshop on Internet Server Performance, 1999.

[250] InfoLibria. Dynacache whitepaper, http://www.infolibria.com

[251] SkyCache, http://www.skycache.com/

[252] CacheFlow, http://www.cacheflow.com/

[253] Akamai Technologies, http://www.akamai.com

[254] http://www.savvis.net/

[255] http://www.wamnet.com/news/read_news.phtml?newsid=686

[256] Maven Networks, http://www.maven.net/

[257] Volera, http://www.novell.com

[258] NetScaler, Inc., http://www.netscaler.com/

[259] Redline Networks, http://www.redlinenetworks.com/

[260] BPVN Technologies Corp., http://www.bpvn.com/

[261] IBM Transcoding Solution and Services, White paper, http://www.research.ibm.com/networked_data_systems/transcoding/transcodef.pdf

[262] Han, R., Bhagwat, P., LaMaire, R., Mummert, T., Perret, V., Rubas, J., Dynamic Adaptation In an Image Transcoding Proxy For Mobile Web Browsing, IEEE Personal Communications, December 1998, pp. 8-17. http://www.cs.colorado.edu/~rhan/Seminar120898.PDF

[263] R. Mohan, J. R. Smith and C. S. Li., Adapting multimedia Internet content for universal access, IEEE Transactions on Multimedia, 1(1):104--114, March 1999.

[264] J. R. Smith, R. Mohan and C. S. Li, Transcoding Internet content for heterogeneous client devices, Proceedings of IEEE International Conference on Circuits and System. May, 1998.

[265] Web Sphere: Transcoding publisher, http://www-3.ibm.com/software/webservers/transcoding/

[266] http://www.filenet.com/

[267] http://www.WebSiteOptimization.com/

[268] http://www.glostart.com/webtrimmer/webtrimmer.html

[269] http://www.hypnotext.com/

[270] http://www.badblue.com/

[271] Ian Marshall and Chris Roadknight, Linking cache performance to user behaviour, In proceedings of 3W3Cache Workshop, Manchester, June 1998.

[272] F. Bonchi, R. Fenu, F. Giannotti, C. Gozzi, G. Manco, M. Nanni, D. Pedreschi, C. Renso, S. Ruggieri, L. Sannais, Adaptive Web Caching Using Decision Trees, SIAM workshop on Web Mining, Chicago, 2001

[273] Robert Buff, Arthur Goldberg, and Ilya Pevzner, Rapid, Trace-Driven Simulation of the Performance of Web Caching Proxies, Submitted to the Workshop on Internet Server Performance, 03/9/98

[274] C. Lindemann, A. Reuys, and M. Reiser, Modeling Web Proxy Cache Architectures, Proc. of 10th GI/ITG Special Interest Conference MMB'99, Trier, September 1999.

[275] Vakali A., An evolutionary scheme for Web Replication and Caching, 4th International Web Caching Workshop, San Diego, USA, March 31-April 2, 1999.

[276] National Lab of Applied Network Research (NLANR) sanitized access log, ftp://ircache.nlanr.net/Traces/

[277] Iyengar A., Challenger, J., Data Update Propagation: A Method for Determining How Changes to Underlying Data Affect Cached Objects on the Web, IBM Research Report RC 21093(94368), February 1998.

[278] J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, The Web as a graph: Measurements, models and methods, Invited survey at the International Conference on Combinatorics and Computing, 1999.

[279] Colin Cooper and Alan Frieze, A general model of web graphs, Proceedings of ESA, pages 500--511, 2001.

[280] Colin Cooper and Alan Frieze, Crawling on web graphs, Proceedings of the 34th Annual ACM Symposiuim on Theory of Computing, 419-427, (2002).

[281] Paolo Boldi and Sebastiano Vigna, The WebGraph framework I: Compression techniques, Technical Report 293-03, Universit di Milano, Dipartimento di Scienze dell'Informazione, 2003.

[282] Paolo Boldi and Sebastiano Vigna, The WebGraph Framework II: Codes For The World-Wide Web, 2003

[283] Sriram Raghavan and Hector Garcia-Molina, Representing web graphs, In Proceedings of the IEEE International Conference on Data Engineering (ICDE03), March 2003.

[284] GNU wget, http://www.gnu.org/software/wget/wget.html

[285] pavuk, http://www.idata.sk/~ondrej/pavuk/index.html

[286] LZ77, http://www.stanford.edu/~udara/SOCO/lossless/lz77/

[287] LZW, http://www.dogma.net/markn/articles/lzw/lzw.htm

[288] Huffman Compression Algorithm, http://www.stanford.edu/~udara/SOCO/lossless/huffman/index.htm, http://www.howtodothings.com/showarticle.asp?article=313

[289] JavaScript Guide, http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/

[290] Chi-Hung Chi, Xiang Li and K-Y. Lam, Understanding the Object Retrieval Dependence of Web Page Access, In Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01), Fort Worth, Texas USA , October 2002. IEEE.