ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.2, February 2009, pp.469−476
doi: 10.3724/SP.J.1001.2009.00553
© by *Institute of Software, the Chinese Academy of Sciences*. All rights reserved.

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

FDE:一种有效的动态网页传送方法*

古志民+, 马俊昌, 程慧芳

(北京理工大学 计算机科学技术学院,北京 100081)

FDE: A Scheme for Efficient Delivery of Dynamic Web Pages

GU Zhi-Min+, MA Jun-Chang, CHENG Hui-Fang

(Department of Computer Science and Engineering, Beijing Institute of Technology, Beijing 100081, China)

+ Corresponding author: E-mail: zmgu@x263.net

Gu ZM, Ma JC, Cheng HF. FDE: A scheme for efficient delivery of dynamic Web pages. *Journal of Software*, 2009,20(2):469−476. http://www.jos.org.cn/1000-9825/553.htm

Abstract: Based on the previous work on automatic detection of shared fragments, this paper proposes a fragment-based delta encoding approach denoted FDE. A feature of FDE is that it can reuse similar content at fragment granularity, and can be deployed transparently to the existing WWW infrastructure. Experiments on 16 popular Web sites show that FDE can provide more bandwidth savings and latency reduction than existing solutions.
Key words: fragment-based delta encoding; ESI; delta encoding; Web caching; dynamic content caching

摘 要: 在作者的片段自动探测算法基础上,提出了一种基于网页片段的Delta编码方法FDE,它能够重用片段和相似片段内容,并且也能透明地部署在现有的万维网基础设施上.实验结果显示,与传统方法相比,FDE 方法能够有效地提高带宽利用率,同时可进一步减少页面传送的延迟.
关键词: 基于片段的 Delta 编码;ESI;Delta 编码;Web 缓存;动态内容缓存
中图法分类号: TP393 **文献标识码**: A

1 Introduction

In the last few years, dynamic and personalized Web pages have increasingly dominated current-day WWW traffic. However, traditional Web caching is applicable only to static pages. To efficiently deliver dynamic pages, a number of techniques have been proposed, among them, ESI (Edge Side Includes)[1] and CDE (Class-based Delta Encoding)[2] stand out. ESI is a specification for an XML-based language that is used to enable assembly of dynamic Web pages from page fragments. ESI relies on the fact that dynamic pages are mostly constructed from the same reoccurring fragments. By caching at fragment granularity, ESI achieves great reuse of content[3]. However, ESI is not transparent to Web developers. Besides, ESI can only reuse *identical* fragments, and can not reuse *similar*

* Supported by the Key Studies Foundation of Beijing and MOE-INTEL-08-10 (北京市重点学科基金和教育部-英特尔信息技术专项基金)

Received 2006-08-10; Accepted 2007-11-20

fragments. There are many similar fragments in Web pages generated by database-driven Web sites. Reusing them through DE (delta encoding)[4] can provide more benefits. In DE, instead of transferring the current complete document, a delta is computed representing the changes compared to some old version. The full document is regenerated at the client. DE can reuse similar documents. Besides, it can be deployed transparently to the existing WWW infrastructure. However, the basic DE scheme is not scalable due to the requirements to store enormous base files[2]. CDE provides a solution to the scalability problem. The idea is to group documents into classes, and store one document per class. However, CDE reuses at *document* granularity and cannot fully exploit redundancy from all documents. For example, if a shared fragment is part of the base files of many classes at a Web site, CDE will require a separate retrieval for each changed base file.

In this paper, based on our previous work on automatic detection of shared fragments[5], we propose a novel approach — FDE (fragment-based delta encoding), which combines the benefits of both ESI and CDE, i.e., reusing at fragment granularity, reusing similar content, keeping transparent to Web developers and the existing WWW infrastructure. The rest of this paper is organized as follows: Section 2 describes our approach in detail. Section 3 evaluates FDE. We conclude in Section 4.

2 Design and Implementation

This section first gives the basic idea of FDE, and then describes it in detail.

2.1 Basic idea

In CDE, there is a delta-server located before the Web server. The delta-server serves as a proxy for client access to the Web server, forwarding the requests from clients to the Web server, and applying DE to responses from the Web server to the clients. If the delta-server holds the up-to-date reoccurring identical and similar fragments and the information about which fragments are referenced by each resource. When the Web server receives a request, it produces the complete page, and delivers it to the delta-server. The delta-server encodes the current page as a delta from the fragments referenced by the requested resource, and sends the delta to the client with the identifiers of the referenced fragments. The client obtains the referenced fragments either from its own cache or, in case of a client-cache misses, from intermediary proxy caches or the delta-server. The client then combines the deltas with the referenced fragments to reproduce the full result page. The above description raises some questions. For instance, how does the delta-server know which fragments referenced by a resource? How does the delta-server encode the current snap-shot of a resource? How can the client understand the encoded content? In the remainder of this section, we answer these questions.

2.2 Automatic detection of shared fragments

We deduce the shared fragments and the mapping information between Web resource and fragments from the generated Web pages. The dynamically created Web pages served by the Web server are logged by the delta-server. Another special machine takes the Web pages from the delta-server as input, analyzes the Web pages, extracts shared fragments, records the information about which fragments are included by each resource and outputs them to the delta-server. The detailed description of the process is described in our previous work[5]. Due to space limitation, we will not duplicate the content here. The shared fragments detection is done periodically to keep the information up-to-date.

2.3 Encoding

Assume the delta-server has owned the fragments and the mapping information, the next question is, given a current snapshot of a resource, how to encode it? First, we combine the fragments frequently referenced by this resource to form a larger file, which is used as the base file. Then we apply delta encoding to the base file. The key point in our scheme is that the base file is identified using the URL list of the contained fragments, rather than one URL. So clients and proxy caches can request and cache the fragments individually. After encoding, the delta-server owns a delta and the URL list of the referenced fragments. Next question is how the delta-server should revise the response so that the client can understand it.

2.4 Modifying response

How to modify a response depends on whether client browser can be modified to supports FDE directly. If client browser can be modified, then we can easily add and extend some HTTP headers to support FDE. Otherwise, we modify the response to utilize JavaScript to automatically support FDE. We discuss the two cases in turn.

2.4.1 Extended protocol

The "Accept-Encoding" request header defined by HTTP provides a means for a client to indicate which content-encoding it will accept. We extend this header to declare the acceptable FDE formats, e.g., "Accept-Encoding: fde-zdelta" means that the client understands FDE, and the DE format can be zdelta[6]. After encoding, the delta-server needs to tell the client the response is delta-encoded. We extend the "Content-Encoding" header to achieve this, e.g., "Content-Encoding: fde-zdelta" means the response is delta-encoded in zdelta format. Besides, the delta-server needs to inform client the referenced fragments. We introduce the "Base-Frags" header to achieve this, e.g., "Base-Frags: /frag/f1 /frag/f2" means the referenced fragments can be retrieved using the relative URL "/frag/f1" and "/frag/f2".

2.4.2 Transparent deployment

If client browser cannot support FDE directly, we revise the response to utilize JavaScript, which is virtually universally enabled, to automatically download referenced fragments, process them to reproduce the original page, and tell the browser to display the restored page. The response body follows the following framework:

```
〈html〉
  〈body id="body"〉
    〈p id="delta" style='display:none'〉delta encoded content〈/p〉
    〈p id="frag" style='display:none'>referenced fragments URL list 〈/p〉
    〈script language="javascript" src="/fde.js"〉〈/script〉
  〈/body〉
〈/html〉
```

The delta and referenced fragments URL list are embedded in two "p" elements. After receiving this response, client browser will not show the delta and fragments URL since the "style" attribute value of both "p" are 'display:none', which means not showing the embedded content. Client browser will download the referenced JavaScript file "/fde.js" and invoke the first line code in that file. This file implements the page assembler. The assembler uses the document.getElementById("delta") statement to obtain the delta encoded content, and document.getElementById("frag") statement to obtain the referenced fragments URL list. For each fragment URL, the assembler uses the XmlHTTPRequest[7] object to retrieve the fragments. Detailed usage of XMLHttpRequest is skipped due to space limitation. After all the fragments are available, the assembler constructs the base file, applies delta to restore the original page, and assigns the page to document.getElementById("body").innerHTML, which causes the browser to show the original page.

The assembler script (fde.js) is generic for all FDE content. Thus once a client downloads it, it remains in its cache and is invoked locally. This is akin to installing a piece of software on the client except this software is installed transparently the first time it is used.

2.5 Techniques for high performance

By avoiding transfer of redundant fragments, FDE can reduce overall bandwidth requirements. However, the request for one page now may involve multiple requests, and some online computations are also introduced. So, the retrieval latency may actually increase. This section discusses some implementation techniques for high performance.

As the delta-server processes a request for a dynamic page, it first forwards the request to the original server. During the waiting time for the response, the delta-server further processes the request and constructs the base file. This overlap of waiting and processing makes the process time imperceptible to clients. Furthermore, to accelerate the processing, the mapping information between fragments and resource is stored in a hash table in memory. Besides, the frequently used fragments and base files are also cached in memory, and do not need to be loaded from disk or constructed every time.

The delta-server can start encoding before receiving the complete original response, and can start sending part of the encoded page before the finish of the encoding. This overlapping of transfer and communication can reduce the user-perceived latency introduced by encoding.

If HTTP/1.1 persistent connection and pipeline are supported by clients, proxy caches and delta-server, the request and response for an encoded page and its referenced fragments can be pipelined over a connection. A client can make requests for fragments as long as the “Base-frags” header is received without waiting for the response body. Besides, the client can make multiple requests for the fragments without waiting for each response. If persistent connection is not supported by some participants, multiple threads can be used.

Shared fragments can be compressed in advance, since the online computation involved is only decompressing, which is typically very fast. The delta-serer holds both the plain and compression version of a fragment. When constructing a base file, the plain version is selected. When satisfying a request for a fragment, the compression version is selected.

2.6 Advantages and limitations

In summary, our approach has the following advantages:

1) It can achieve great bandwidth savings through fragment granularity similarity reusing. For a new request, if the response contains an identical fragment that has been transferred before, then its content need not be transferred again; if a similar fragment has been transferred before, then only a delta needs to be transferred.
2) Since frequently referenced fragments are typically cached at clients and proxy caches, and we can use techniques described in the last section to implement efficiently, FDE can reduce retrieval latency.
3) It can ensure strong consistency since all the requests will be processed by the original server. Delta-Server, proxy caches and clients only change the encoding format and do not affect the semantic meaning. Strong consistency is required in many Web sites.
4) The fragments in FDE have high reusability and small storage requirements, because a fragment is often shared among many Web pages and it needs to be stored only once.
5) It can be deployed transparently to the current WWW infrastructure. In section 2.4.2, we have shown that FDE can be deployed transparently to client browser. It’s easy to see that FDE is transparent to the original Web server. FDE can be applied to any Web servers and Web applications without any changes on them. FDE is also transparent to proxy caches. FDE requires no changes on proxy caches and can

utilize them to cache fragments.

Like CDE, the primary limitation of FDE is that it needs online DE computation. The DE time is insignificant from client’s perspective; however, since DE is CPU-intensive, it has obvious negative effect on throughput. However, CPU is cheap in comparison to the cost of access links. In practice, it is very common that the bottleneck resource at a Web server is the access link out of the Web site and not the CPU. Besides, we can also implement a self-adaptive policy in the delta-server that can stop modifying response in case of the delta-server is overloaded or the network bandwidth is high enough.

3 Evaluation

In this section, we experimentally evaluate FDE. We first introduce the experiment environment, method and data sets, then report the results on bandwidth requirement and performance speedup of FDE and other three schemes (compression, ESI and CDE), and finally we study the computation and storage overheads introduced in FDE.

3.1 Experiment environment, method, and data sets

We have three machines to act as client, server proxy and Web server respectively, the configurations are as follows. Client: Intel Celeron CPU 1.2GHz, 128MB RAM, Linux 2.4. Server Proxy: Intel Celeron CPU 2.4GHz, 512MB RAM, Linux 2.4. Web Server: Intel Celeron CPU 1.0GHz, 128MB RAM, Linux 2.4. The three machines are in the same 100Mbps network. We used a WAN emulator - NIST[8] to emulate the WAN link between client and server proxy. The Web server software used is Apache 2.0. We implemented a prototype of the server proxy software, which can be switched easily to test various schemes. For comparison between ESI and other schemes, the ESI encoded pages are compressed in the server proxy before sent out. The delta encoding and compression algorithm used is zdelta and gzip respectively. The client software is a program written by us, which reads a URL list, for each URL, retrieves the content and accounts the latency and consumed bandwidth.

An input to the above experiment system is a data set consisting of dynamic Web pages downloaded from a real Web site. Given the pages, we detect the shared fragments, and record the mapping information between fragments and pages into a file. In our detection, a fragment is detected only when it appears at least ten times and its size is not less than 512 bytes. Based on the original data set, we generate two new data sets: ESI encoded page set and base file data set. The base file set is used in CDE. All the fragments, base files and mapping file are stored at the server proxy. Both the original pages and ESI encoded pages are put in the Web server as static HTML pages. Static pages do not need extra processing time. However, this time does exist at real systems. To model this latency, we make the server proxy delay a typical time (100ms in our study) before further processing a response from the Web server.

We downloaded 16 large sets of Web pages from 16 popular Web sites as the experiment data sets. The tool to download the pages is GNU Wget (http://www.gnu.org/software/wget/). Of the 16 Web sites, 8 of them are selected from the top 50 popular Web sites in China (reported in http://www.cwrank.com), and the other 8 sites are selected from the top 50 popular Web sites in US (reported in http://www.comscore.com/metrix/). In our selection, we try to cover various types of Web sites that primarily provide dynamic Web pages including portal, e-commerce, entertainment, etc. Table 1 lists the information of the selected Web sites.

3.2 Results on bandwidth requirement and performance speedup

For each of the 16 data sets, we obtain the retrieval time and total bytes transferred in the WAN link to download all the pages in five cases: Unmodified, compression, ESI, CDE and FDE. This section reports the results on relative bandwidth requirement and performance speedup of compression, ESI, CDE and FDE when compared with the unmodified system.

Figure 1 shows the relative bandwidth requirement of the four schemes. As shown in this figure, although for some sites (e.g., classmate), the bandwidth requirement in ESI and CDE is close to that in FDE, FDE consistently requires the least bandwidth for all the Web sites.

Table 1 Name, page number, average size and compression ratio of selected sites

Name of Web Sites	Alias	Page number	Average size (KB)	Average compression ratio (%)
www.51job.com	51job	17540	20.700	29.22
www.163.com	163	2755	26.543	31.49
china.alibaba.com	alibaba	7278	43.869	21.71
www.bokee.com	bokee	12350	22.819	31.86
www.careerbuilder.com	caree	2558	69.787	17.02
www.cctv.com	cctv	6283	17.861	31.23
www.cheaptickets.com	ticket	9675	52.735	25.16
www.classmates.com	classmate	11420	21.029	29.29
www.dangdang.com	dangdang	5618	58.980	20.50
www.hollywood.com	hollywood	7509	36.518	20.66
www.infospace.com	infospace	3837	28.030	17.69
www.orbitz.com	orbitz	13070	45.173	24.68
www.pcauto.com.cn	pcauto	13851	32.996	24.43
www.pconline.com.cn	pconline	18196	45.676	22.01
www.overstock.com	overstock	10422	63.149	17.33
www.walmart.com	walmart	6253	69.506	15.96

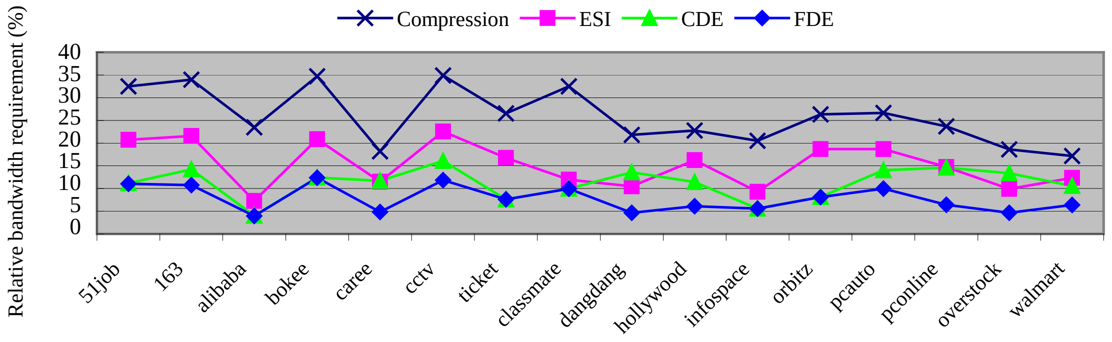


Fig.1 Relative bandwidth requirement for 16 sets of Web pages

Figure 2 shows the performance speedup of the four schemes when latency and bandwidth in the WAN link is set to 100ms and 100Kb/s respectively. As shown in this Figure, similar with Fig.1, although the performance of ESI and CDE is close to FDE for some Web sites, FDE consistently provides the best performance for all the Web sites.

We examine how the latency and bandwidth in the WAN link affects the results. We fix latency at 8/100/200/1000ms, and study the average speedup of the 16 Web sites when bandwidth varies from 16Kb/s to 2Mb/s, as is shown in Fig.3. It is clear that FDE is consistently better than other schemes, although the benefit is not very obvious in some cases.

3.3 Computation and storage overheads

In our experiments, the computation time needed to encode a page in server side is hard to be perceived by a user, e.g., it only takes 31.14ms even for the most time-consuming Web site (caree). The computation time consumed at client side is negligible, e.g., it only takes 2.43ms even for the most time-consuming Web site (caree) in FDE.

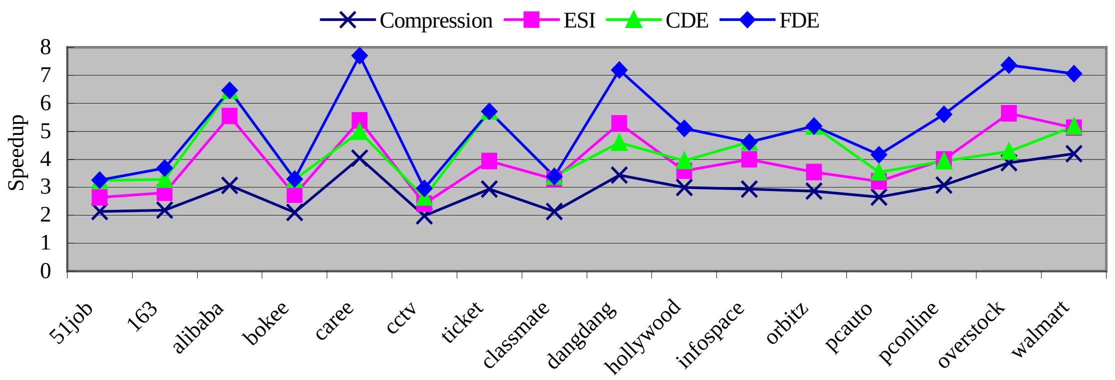
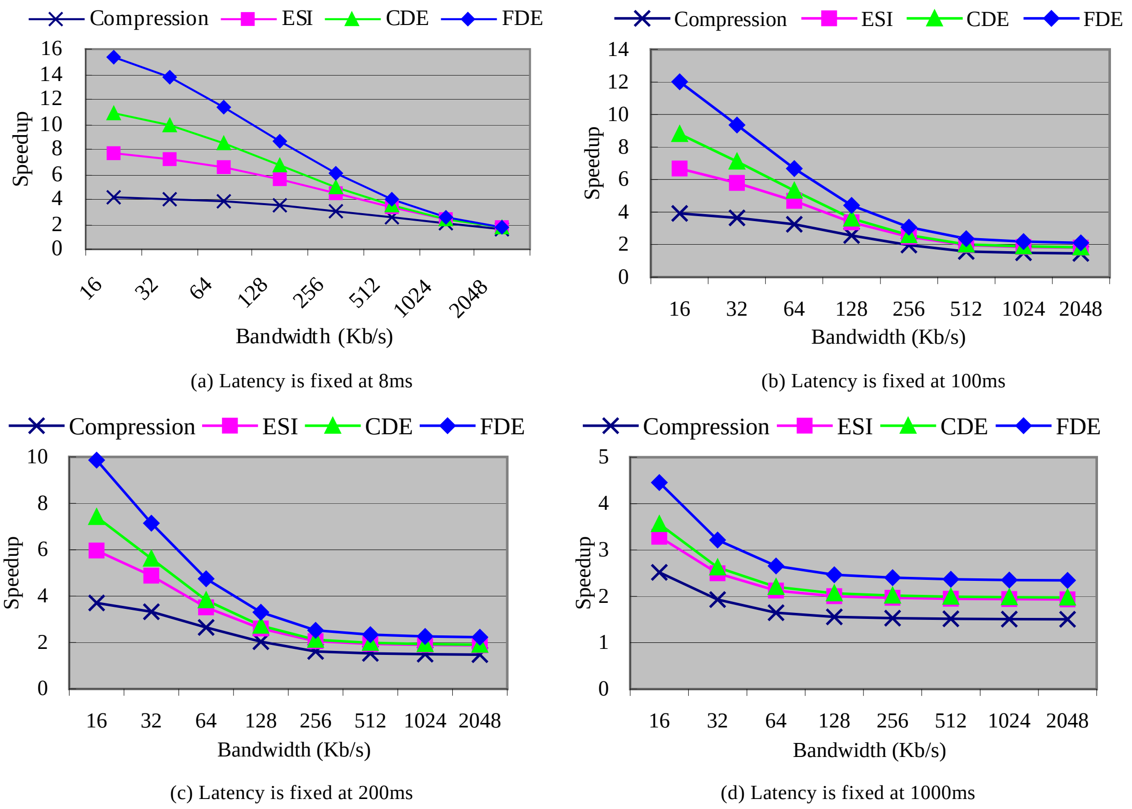


Fig.2 Speedup of 16 data sets when latency is 100ms and bandwidth is 100KB/s



(a) Latency is fixed at 8ms

(b) Latency is fixed at 100ms

(c) Latency is fixed at 200ms

(d) Latency is fixed at 1000ms

Fig.3 Effects of latency and bandwidth on performance speedup

FDE requires client and sever proxy to store fragments, to evaluate the storage overheads, we examine the total size of fragments versus the total size of original pages. We find this ratio is very low for all the Web sites. On average, it is only 1.06%. Even in the worst case, it is only 3.71% (163).

FDE also requires server proxy to store the mapping information between Web resource and referenced fragments in memory. Here, it is worthy to be noted that a Web resource is not an instance page, but typically is a Web application that generates many homogeneous pages. For a medium Web site, suppose the number of Web resources is 1000, each resource references an average of 5 fragments. Each resource can be identified by an MD5 value (16 bytes). Each fragment can be identified by an integer (4 bytes). So, plus two pointers (8 bytes) used in hash table, each record needs 44 bytes. Thus, the total hash table needs only about 43KB; even there are 10000 Web resources, the size needed is only about 430KB. The frequently used fragments can also be cached in memory. For a typical medium Web site, suppose there are 1000 popular fragments, the size of each fragment is 16KB, then the total size needed is only about 16MB; even there are 10000 fragments, the size needed is only about 160MB, which is easy to be satisfied for a current server machine.

In summary, the computation and storage overheads introduced in FDE are acceptable.

4 Conclusion and Future Work

This paper proposes an efficient approach FDE to deliver dynamic Web pages. FDE combines the benefits of both ESI and CDE; it can reuse similar content at fragment granularity and at the same time can be deployed transparently to the existing WWW infrastructure. The main limitation of FDE is it needs online delta encoding, which has negative effects on the server side throughput. So the main application of our scheme is to Web sites that primarily provide dynamic Web pages and the server side CPU resource is not a bottleneck or can be scaled easily. In the future, we plan to deploy our scheme in some real Web sites and evaluate it in the real environments.

Acknowledgement We thank Prof. Dan Meng for discussing some issues about this paper.

References:

[1] ESI: edge side includes. http:// www.w3.org/TR/esi-lang

[2] Psounis K. Class-Based delta-encoding: A scalable scheme for caching dynamic Web content. In: Proc. of the IEEE ICDCS. Los Alamitos: IEEE Computer Society, 2002. 799−805.

[3] Naaman M, Garcia-Molina H, Paepcke A. Evaluation of ESI and class-based delta encoding. In: Proc. of the WCW 2003. New York: IBM T.J. Watson Research Center, 2003. http://www.iwcw.org/2003/

[4] Mogul JC, Douglis F, Feldmann A, Krishnamurthy B. Potential benefits of delta encoding and data compression for HTTP. In: Proc. of SIGCOMM’97. New York: ACM Press, 1997. 181−194.

[5] Gu ZM, Ma JC. Automatic detection of shared fragments in large collections of Web pages and its applications. Journal of Algorithms & Computational Technology, 2007,l1(2):215−250.

[6] Trendafilov D, Memon N, Suel T. zdelta: An efficient delta compression tool. 2002. http://cis.poly.edu/tr/tr-cis-2002-02.pdf

[7] XMLHttpRequest. http://www.w3.org/TR/XMLHttpRequest/

[8] Carson M, Santay D. NIST Net-A Linux-based network emulation tool. http://snad.ncsl.nist.gov/nistnet/nistnet.pdf

GU Zhi-Min received his Ph.D. degree in Computer Science from Xian Jiaotong University in 1997. Now he is a professor of Computer Science at Beijing Institute of Technology. His research areas are distributed and Internet systems, computer cluster architecture and science computing.

CHENG Hui-Fang received her Ph.D. degree in Computer Science at Beijing Institute of Technology in 2006. Her research areas are Internet systems and P2P technologies.

MA Jun-Chang received his Ph.D. degree in Computer Science at Beijing Institute of Technology in 2006. His research areas are Internet systems, Web technologies, cluster computing, computer architecture and operating systems.