Providing Adaptive Dynamic Web Content in Mobile Environment

Zhigang Hua[2], Xing Xie[1], Hanqing Lu[2], Wei-Ying Ma[1]

[1]Microsoft Research Asia
5/F Sigma Center, No. 49 Zhichun Road
HaiDian District, 100080 Beijing, China
+86-10-62617711
{xingx, wyma}@microsoft.com

[2]Institute of Automation
Chinese Academy of Sciences
P.O. Box 2878, 100080 Beijing, China
+86-10-62542971
{zghua, luhq}@nlpr.ia.ac.cn

ABSTRACT

Dynamic content has dominated the Web nowadays. Although much work has been carried out to adapt web pages for displaying on small-screen mobile devices, none of them has taken the prevalent dynamic web page into a special consideration. Dynamic pages have to be renewedly generated and transmitted over wireless networks even when a small portion changes. Since the adaptation also has to be performed on whole pages, this leads to considerable user-perceived delay. In this paper, we propose a unified framework based on fragment-based technologies which have been commonly applied in web caching. In our framework, a novel page adaptation method called constrained page splitting is used to improve readability of dynamic pages, then split results rather than original contents are saved to cache in fragment granularity. Our approach can simultaneously improve readability and user-perceived delay when browsing dynamic web pages on mobile devices. In the evaluation, we use a representative Web benchmark to measure the performance of our framework. The experimental results point out that our approach is effective in improving the mobile browsing experience.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications – *Information browsers;* H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia – *Navigation;* H.3.4 [**Information Storage and Retrieval**] Systems and Software – *distributed systems, performance evaluation (efficiency and effectiveness);*

General Terms

Algorithms, Measurement, Experimentation

Keywords

Constrained page splitting, dynamic web page, fragment caching, Fragment Cache Hit Ratio, readability, user-perceived delay, mobile browsing experience, mobile device, web page adaptation.

1. INTRODUCTION

Dynamic web pages become increasingly dominant in the Web nowadays. On the one hand they provide more personalized and flexible services to users, but on the other hand they delay response time and increase web server's processing load. Presently, mobile devices are becoming more and more popular, such as Handheld PCs, SmartPhones and PDAs. Though it is convenient to access the Internet from the portable mobile devices even when users are on the move, there are still many hurdles needed to be crossed, such as small display, limited bandwidth and poor computing power. Small display will greatly degrade the readability of web pages which are usually too large for small-screen mobile devices. Besides, limited bandwidth and poor computing power will increase user-perceived delay. Therefore, a challenging problem is how to efficiently improve the browsing experience of prevalent dynamic web pages on mobile devices.

Since currently most web pages are mainly designed with desktop computers in mind, readability is usually degraded because they are too large to display on small-screen mobile devices. Many adaptation methods [3, 4, 5, 7, 8, 11, 13] have been proposed to adapt web pages to be tailored for small-screen mobile devices. Among them, page splitting is a basic technique to divide a web page into tailored blocks, which can fit into the small display size in mobile devices. In prior work, Chen[5] proposed an efficient page splitting algorithm based on analyzing the position and shape of HTML elements. However, the main difficulty of page splitting algorithms is how to effectively extract tailored blocks from low level HTML tags without destroying page structure and causing information losing. In addition, user-perceived delay is also largely increased since these adaptation methods are usually time-consuming. According to our practices, adaptation costs on mobile devices can't be ignored for the poor computing power of mobile devices.

Speeding up the delivery of dynamic pages remains a hot topic in recent years. Conventional approaches which cache in page granularity can not work effectively for dynamic web pages, because they should be renewedly generated and transmitted even when a small portion within them changes. A few other caching strategies have been proposed to address this problem [1, 6, 9, 10, 14, 15, 16]. Among these methods, encoding web pages with fragments and caching in fragment granularity proved to be an effective way, which can accelerate the delivery of web pages and alleviate web server load. Extensive investigation of different offloading and caching options has been made in [15]. An approach to

enable fragment caching at the network edge was proposed in [16], which requires only simple modifications to existing applications. However, it can't benefit mobile web browsing, because the download time of web pages is determined by the last mile, i.e. the slow dial-up link that mobile devices use to access the Internet. Based on this fact, [10] proposed to push ESI (more detailed information can be referred to [6]) to the ultimate network edge – the client side, which can accelerate download time of web pages for clients. While for mobile devices, it still does not suffice to improve user browsing experience due to the poor readability of large web pages on the small-screen mobile devices. If adaptation is then performed independently, valuable semantic information indicated by fragments has not be used, and user-perceived delay will still be deferred by the time-consuming adaptation cost since these caching methods have not considered to cache adaptation results.

In our proposed framework, we comprehensively integrate caching and adaptation together. In the framework, constrained page splitting is presented to improve the readability of dynamic web pages by utilizing semantic information indicated by fragments; then split results rather than original contents are saved to cache in fragment granularity, which is an extension to current caching methods. As a consequence, split results of cached fragments can be directly fetched from cache to avoid the repeated transmission and adaptation costs in later requests. Afterwards, a request to the same page or other pages containing the same fragments will benefit from our approach. In the evaluation, a famous Web application, namely IBuySpy, is used as a representative benchmark to measure the performance of our framework. Experimental results point out that our approach is effective in improving mobile browsing experience.

The main contributions of this paper include:

- User-perceived delay and page readability are two most important factors for mobile web browsing. Our framework integrates caching and adaptation together to deal with both problems simultaneously.
- We propose constrained page splitting algorithm to utilize the higher level semantic information indicated by fragments to improve the adaptation performance, instead of analyzing only low level HTML tags in conventional ways.
- The details of our prototype implementation are provided. Our implementation experience will be useful for readers who implement other functionality applying fragment-based technologies or web page adaptation.

The rest of this paper is organized as follows. Section 2 introduces fragment-based technologies. We propose our unified framework in Section 3. In section 4, we introduce constrained page splitting method. Section 5 presents our approach on how to save split results to cache. In Section 6, we compare three different mainstream deployment schemes and conclude that deploying our framework at client side is the best approach. We provide implementation details in Section 7. Our calculation method of Fragment Cached Hit Ratio (*FCHR*) is presented in Section 8. In section 9, we give the experimental results and discussions. Finally, we conclude this paper and introduce our future work in Section 10.

2. FRAGMENT-BASED TECHNOLOGIES

Fragment caching is a promising technology to deal with the problem of delivering dynamic web pages recently. According to current existing fragment-based technologies such as ESI [6] and Proxy+ [16], a web page is encoded with additional semantic tags which indicate the cacheable characteristics of its included fragments. Each fragment is treated as a separate object in a web page, and has its own cache and access profile which may be set in a configuration file. For instance, content providers may want to cache the template of a page for several days, but only cache a particular fragment (such as a stock quote) for a matter of seconds or minutes. If a web page composed of fragments is considered as an individual whole, its lifetime is limited to the minimal lifetime of fastest changing fragment inside.

As for an example of a web page composed of cacheable fragments, consider a personal home page, which is shown in Figure 1 below. In this paper, an HTML unit is referred to an HTML content portion within a web page, while a fragment can be deemed as an HTML portion with specified cacheable properties. In the figure, "activity" fragment *f3* and "career" unit *c1* are included in "news" fragment *f2*; *f2* and "diary" fragment *f1* are included in the template fragment *f0*, which also includes "favorite" unit *c2*. The fragments within the page like *f0, f1, f2* and *f3* have their own lifetimes which range from one day to more than a hundred days. This example also demonstrates an issue that a fragment can include one or more other fragments, which means that fragments can be recursively defined. For instance, the fragments *f1* and *f2* are included by the outer template fragment *f0* which maintains the template of the page, and *f3* is also encircled by *f2*. Therefore, a space occupied by a fragment may be overlapped by other fragments. This will add complexity to our page adaptation algorithm, as we will discuss in detail later.

In this case, when there is a client request to a web page encoded with cacheable fragments, web server will examine the cache status of each fragment within the requested page. Only changed fragments need to be generated by web server. After fetching other unchanged fragments from cache, a whole page is then completely assembled to answer the client request. As a consequence, this approach can accelerate response time and lighten web server load by processing only changed fragments, instead of whole pages in conventional approaches. The results in [15] show that under typical user browsing patterns and network conditions, 2~3 folds of latency reduction can be achieved. Furthermore,

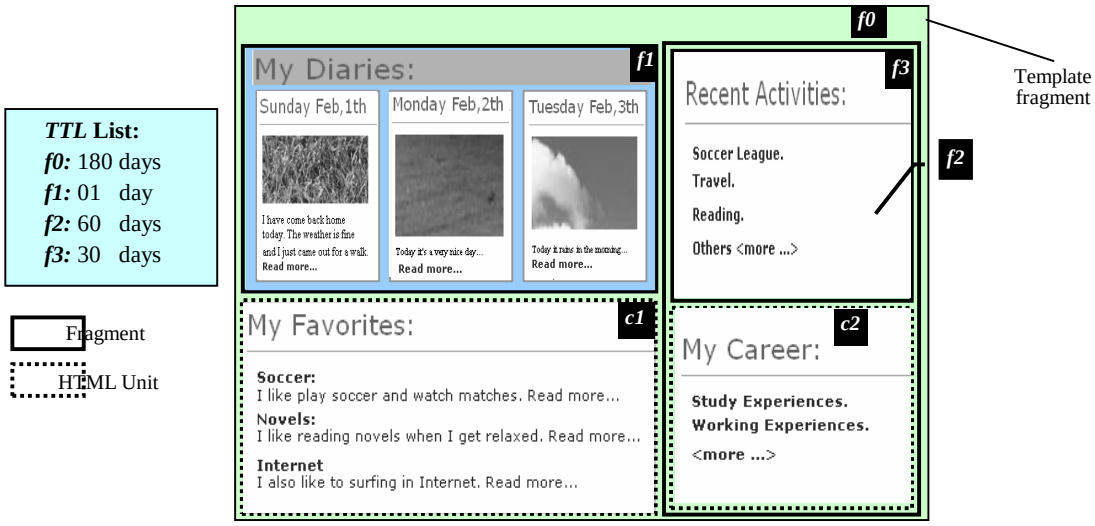


Figure 1. Two fragments named *f1* and *f2* are recursively contained in the outer template fragment *f0*, which includes a HTML unit named *c2*. *f2* includes a fragment *f3* and a HTML unit *c1*.

over 70% server requests can be processed by fetching from cache, resulting significant server load reduction.

Generally, current fragment-based technologies are mainly focused on web caching strategies. Actually, a fragment is a relatively independent unit containing semantic information within a web page. Based on this, in this paper we propose constrained page splitting method to use semantic information indicated by fragments to improve page adaptation performance. The details of our approach will be introduced later. In this paper, we assume that dynamic web pages mentioned later are acquiescently thought to be encoded with cacheable fragments. Generally, *fragment* mentioned later is not limited to a specified fragment definition, for it is generic enough to be applicable to various fragment-based technologies.

3. A UNIFIED FRAMEWORK

User-perceived delay and page readability are two key factors determining mobile browsing experience. In this paper, our goal is to improve both.

Currently, web servers in the Internet commonly do not provide page adaptation service to improve web page readability for small-screen mobile devices. Assume a scenario where a user browses a dynamic web page on a small-screen mobile device capable of page adaptation functionality. The browsing process goes like this. First, the requested dynamic web page is generated by remote web server and then transmitted over wireless network to reach the mobile device. Second, local page adaptation process is invoked to adapt the received page to fit the small screen in the mobile device. From the process we can conclude that user-perceived delay of dynamic web pages is heavily increased, and page adaptation also imposes much processing load on mobile devices.

It is known to us that page adaptation targets at improving web page readability, and caching targets at accelerating page delivery. To improve mobile browsing experience of dynamic web pages, we found that adaptation and caching should work cooperatively. If they are treated separately, it will incur some problems to degrade the overall performance. The first problem is that adaptation may be often repeatedly executed on same pages or similar pages, which can be potentially saved. The reasons are twofold. Firstly, popular pages visited frequently will be repeatedly adapted even within their lifetimes; Secondly, same fragments shared among different pages will be repeatedly adapted too. Another problem is the difficulty for page adaptation process to extract tailored semantic blocks by only analyzing low level HTML tags. This may result that adaptation results are not consistent with user's perception of web page structure, while higher level semantic information indicated by fragments has not been used to improve adaptation effects. Based on these problems caused by treating caching and page adaptation separately, in this paper, we propose a unified framework which comprehensively integrates caching and adaptation together. In this framework, a new page adaptation approach called constrained page splitting is presented to improve the readability of dynamic web pages, then split results rather than original contents are saved to cache in fragment granularity.

By using higher level semantic information indicated by fragments instead of low level HTML tags, the readability of dynamic web pages can be improved because adaptation results processed by our approach can correspond much better to user perception. By caching split results rather than normal contents, only missing fragments within pages need to be transmitted and split, while other cached fragments can be fetched directly from cache to save the repeated processing. Therefore, user-perceived delay of dynamic web

pages can be reduced. Thus, our framework simultaneously improves readability and user-perceived delay. The experimental results demonstrate that our approach is effective in improving mobile browsing experience later.

As shown in Figure 2, our framework contains two main processes. First, constrained page splitting is used to split dynamic web pages to be adaptable for mobile devices (discussed in Section 4). Second, the split results are saved to cache in fragment granularity (described in Section 5).

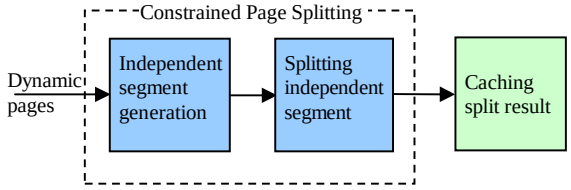


Figure 2. The work flow of our framework.

4. CONSTRAINED PAGE SPLITTING

From Figure 2, we can see that constrained page splitting method is comprised of two steps. First, we analyze the fragment structure within a dynamic web page and generate a set of independent segments. Then each independent segment is split into blocks which can fit well into the small screen of mobile devices.

4.1 Independent Segment Detection

As mentioned in the example (Figure 1), fragments within a page can be recursively defined. Therefore, a fragment can also include one or more other fragments. This adds complexity to adaptation algorithms since the space occupied by recursive fragments is overlapping. Obviously, a splitting algorithm should avoid repeated operations on the overlapping spaces.

In this section, we define an independent segment as a HTML content unit that does not include any fragment within its area. It per se can be a fragment or a HTML unit. As shown in Figure 1, the independent segments should be two fragments *f1* and *f3* and two HTML units *c1* and *c2*. To eliminate fragment overlapping within a dynamic web page, an algorithm to detect all the independent segments in a page will be presented.

4.1.1 Independent Segment

To facilitate the description of our algorithms, we assign five properties to an independent segment as following.

Definition 1: An independent segment within a dynamic web page is assigned five properties:

$$Seg_i = \{(HC_i, SR_i, DF_i, IF_i, IC_i)\}, 1 \le i \le N \quad (1)$$

where

Seg_i,	the i^{th} independent segment within the page
HC_i,	the corresponding HTML content of Seg_i
SR_i,	the split result of Seg_i
DF_i,	the fragment which Seg_i directly belongs to
IF_i,	whether Seg_i is a fragment
IC_i,	whether the split result of *Seg* is cached
N,	the total number of independent segments

Five properties are assigned to an independent segment: *HC*, *SR*, *DF*, *IF* and *IC*. *HC* is used to represent the corresponding HTML content of *Seg*. *SR* stands for the split result of an independent segment. *DF* is defined as the fragment which Seg_i directly belongs to. Especially, if Seg_i is included by multiple upper-level fragments, its *DF* will be the fragment that directly contains it. For instance, though *f3* is included by *f2* and *f0*, its *DF* is *f2*. According to our definition, a *DF* itself should not be an independent fragment. For the root fragment in a page, or called the template fragment, its *DF* will be set to *null* since it is not included by any fragment. For instance, in Figure 1, *DF* of *f1* and *c2* is *f0*, *DF* of *f3* and *c1* is *f2*, and *DF* of *f0* is *null*. *IF* and *IC* are two Boolean variables which indicate whether an independent segment is a fragment and whether its split result has been cached, respectively.

4.1.2 Detection Algorithm

In the algorithm for detecting independent segments within pages requested, we first use a typical HTML parser to generate a HTML DOM (Document Object Model) tree from the web page. Then the DOM tree is traversed from the root node to its leaves until all independent segments have been detected. Figure 3 shows the structure of the HTML DOM tree for fragment *f0* in Figure 1.

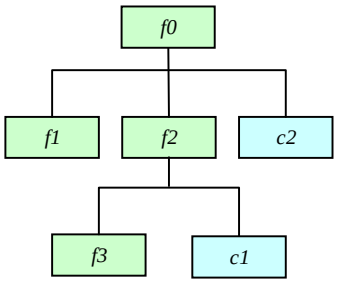


Figure 3. The structure of the HTML DOM tree for the template fragment *f0* in Figure 1.

Actually the detection of independent segments can be considered as a problem to find leaf nodes within a tree. It is based on a depth-first traversal in the HTML DOM tree, which starts from the root node in the tree. We will describe in the following steps on how to detect independent segments and set their properties:

1) For each child node of the current node, go to step 2.

2) Check whether the node contains any fragment within its scope. If it contains fragments, return to step 1. Otherwise, it is considered as an independent segment, perform steps 3–7 to set its properties.

3) *HC* is copied from the content of the node, while *SR* will be generated later by the splitting algorithm in the next section.

4) The value of *DF* can be set to the identifier of the fragment which the node directly belongs to.

5) Examine whether the node is a fragment. If it happens to be a fragment, set its *IF* to 1. Otherwise, set its *IF* to 0.

6) If the node is a fragment, the value of *IC* will be set to 1 when cached or 0 when not cached. Otherwise, continue checking whether its *DF* is cached and set the corresponding *IC* value.

7) Insert the independent segment into a queue.

After using this algorithm, a queue of independent segments will be obtained within a dynamic web page. In the next step, a splitting algorithm will be performed to generate the value of *SR* for each independent segment.

4.2 Splitting Independent Segments

Our goal in this step is to split the independent segments into a series of semantic blocks that can fit into the small screen of mobile devices. Here the relationship of three granularities mentioned in this paper will be explained, they are fragment, segment and block respectively. In term of our definitions, a fragment may include one or more independent segments, and an independent segment may be split into one or more blocks. According to our practices, we found that most independent segments usually can not be further split. Thereby, the time complexity of our page splitting algorithm is greatly reduced.

Our splitting algorithm is a depth-first traversal of the HTML DOM tree based on analyzing the position and shape of each node, which starts from the root node of each independent segment. Note in our splitting algorithm, we use different methods to process missing segment and cached segment.

As for the missing independent segments whose split results have not been cached (i.e. *IC=0)*, the splitting algorithm can be summarized as the following pseudo code:

```
BlockSet = Φ        // the set of split blocks initiated as null
SplitMissSeg (S)   // S is a missing independent segment
{
  For (each child node S_i ∈ S) {
    If (Fit-Display( S_i )==TRUE or
       Not-Splittable( S_i )==TRUE)
          BlockSet = BlockSet ⋃ S_i
    Else  SplitMissSeg ( S_i );
          // Recursively splitting child nodes of S_i
  }
}
```

After being split by our algorithm, each missing independent segment will be split into a set of blocks which can fit into the display size of mobile devices. As an example to split a missing segment, consider the fragment *f1* shown in Figure 1. Figure 4 (a) shows the DOM structure of the fragment *f1* (Figure 1). After handled by the splitting algorithm, *f1* is split into four blocks labeled by numbers from 1 to 4, as is shown in Figure 4(b).

In order to save the split results, in our framework, we insert delimiting tags into the original content to denote the boundary of each block split by our approach. Therefore, later requests can retrieve the blocks directly from the previously inserted tags to avoid repeated splitting. The split results inserted with delimiting tags will be saved to the property of *SR* in each missing independent segment, which will be saved to cache in the latter caching process. Figure 5 shows an example of the split result for *f1*. In the example, four blocks are wrapped by delimiting tags (e.g. <*tag*>) to indicate their boundaries.

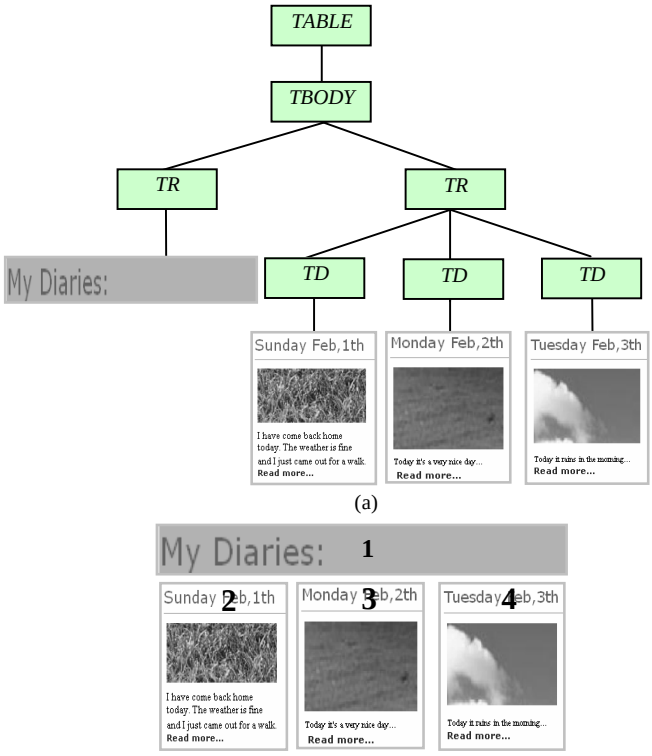


Figure 4. (a) The DOM structure of the fragment *f1*. (b) *f1* is split into four blocks by the splitting algorithm.

```
<TABLE><TBody>
<tag><TR>…</TR></tag>
<TR>
    <tag><TD>…</TD></tag>
    <tag><TD>…</TD></tag>
    <tag><TD>…</TD></tag>
</TR>
</TBody></TABLE>
```

Figure 5. The cached split result of *f1* within which the inserted delimiting tags (e.g. <tag>) denote boundaries of the split blocks.

As for the cached independent segments (i.e. *IC=0)*, using

the inserted delimiting tags which indicate the boundaries of split blocks, we present the following pseudo code to describe the algorithm to split them:

BlockSet = Φ // the set of split blocks initiated as null
SplitCachedSeg (S) // S is a cached independent segment
{
For (each boundary-tagged child node $S_i \in S$*)*
BlockSet = BlockSet $\bigcup S_i$
}

Compared to the splitting algorithm of missing segments, we can see that time complexity of splitting cached segments can be greatly reduced.

5. SPLIT RESULTS CACHING

Current fragment caching strategies only save original contents. Though they can speed up the delivery of dynamic web pages by saving the transmission costs of cached fragments, they still do not suffice to improve mobile browsing experience since readability problem has not been addressed. In our unified framework, the split results rather than the original contents of each fragment are saved to cache. Therefore, in later requests they can be fetched from cache to save both the transmitting and splitting costs. As a result, user-perceived delay can be reduced while simultaneously improving readability.

The technical problem behind is how to efficiently cache the split results of a fragment. As described previously, after splitting, we can get a queue of adapted independent segments which are tagged with block boundary information. Though the split results are in block granularity, the caching should be in fragment granularity to accord with current fragment caching mechanisms. Therefore, we should be able to save all the adapted independent segments within a fragment as a whole to cache. Note that our caching here only operates on missing independent segments, i.e. *IC=0*. For cached independent segments, i.e. *IC=1*, since their split result has been already cached, it's unnecessary to cache them again.

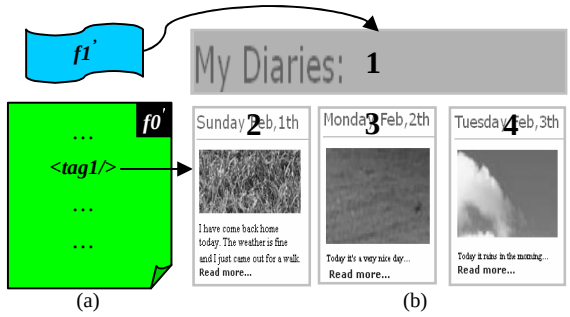


Figure 6. (a) The cached split result of *f0* within which *<tag1>* is inserted to indicate the split result of *f1*. (b) *f1* is split into four blocks according to the inserted delimiting tags.

The caching algorithm is illustrated in the following:

1) For each independent segment which has not been cached (i.e. *IC=0*) in the queue (obtained in the prior step), go to step 2.
2) Check the value of its *IF*. If it is a fragment, save *SR* directly to its cache (e.g. *f1* in Figure 6), and go to step 3. Otherwise, save *SR* to the cache of its *DF*, continue processing the next independent segment.
3) Examine whether its *DF* is null. If its *DF* is not null, insert special tags (e.g. *<tag1>* in Figure 6 (a)) into its *DF* to indicate the split result of the fragment. Otherwise, continue processing the next independent segment.

As for an example, the cached split result of *f1* is shown in Figure 5. Figure 6 (a) shows the cached split result of *f0* within which *<tag1>* is inserted to indicate the split result of *f1*.

When this page is requested later, Figure 6 gives an example on how to split the cached fragment *f1*. When begging to split the page, we start from the outer template fragment *f0*. After fetching its split result from cache, a special tag (e.g. *<tag1>*) is detected which indicates its included fragment *f1*. Then the tag will be replaced with the corresponding cached split result for *f1*. With the delimiting tags previously inserted in the cached split result (e.g. *<tag>* in Figure 5), we can easily detect the boundaries of four blocks to split *f1*, shown in Figure 6 (b).

6. DEPLOYMENT TRADEOFF

As aforementioned, our framework is mainly designed to use constrained page splitting to make a web page tailored for small-screen mobile devices, and then save the split results of missing fragments to avoid repeated processing afterwards. There are a number of possible places to deploy our framework: web server, network edge or client side. We should choose from the three possibilities to select the best scheme to provide a better service to mobile devices.

The most common way is to deploy it at web server. In this case, web server responses a mobile client with an adapted page instead of an original page. By reusing split results of cached fragments within pages requested, the web server can avoid their generation and adaptation costs. But the outbound traffic from the web server has not been reduced since the full web page still needs to be transmitted over the limited bandwidth network that mobile devices connect to access the Internet. Therefore, the user-perceived delay has not been improved in this deployment scheme. What's more, the web server load will also be heavily increased.

The second possibility is to deploy our framework at edge servers which are located at the network edge close to clients. Edge server assembles an original page and then adapts it to return a tailored version to mobile clients. By reducing the bandwidth consumption on the proxy to server link and possibly the web server load, deploying at edge servers seems to be an acceptable strategy. However, in this

case, the delivery of web pages for mobile devices can hardly be accelerated because the bottleneck in the page delivery turns out to be the last mile, i.e. the slow dial-up link. Security issues also become a critical problem since the edge server has the ability to substitute or modify the original content without authorization.

What's more, both schemes discussed above will encounter the problem of caching storage explosion. This is due to the fact that different characteristics of mobile devices like display size will result in different adapted versions of the original web page even for same fragments or pages. This will greatly increase the size of cached data on web servers or edge servers. Situations become even worse when the dynamic pages are personalized for each user, which is very common recently. Therefore, caching storage explosion is a critical problem which restricts the performance of these two deployment schemes. In addition, edge servers or web servers must be smart enough to detect characteristics of client devices to provide the right cached version of adapted pages.

In this paper, we propose to directly push our framework directly to mobile client. In this case, when mobile clients begin to request a fragment-composed web page from a web server, the request header will be modified to download only missing fragments instead of the whole page. Thus, when fragment-aware web server receives such a request, it only returns the missing fragments within the page to clients. By assembling downloaded and cached fragments, a whole page is composed at mobile client. After the assembled page is handled by our splitting method, an adapted page is finished to display on mobile devices. To our knowledge, to deploy at the client side has at least three main advantages:

- The caching storage explosion problem can be avoided for the configuration of a mobile client is usually fixed during a certain period of time.
- User-perceived delay can be accelerated since less data needs to be transmitted over the slow dial-up link of mobile devices.
- The edge server commonly used to speed up the delivery of web pages is not necessary any more for mobile devices. This greatly reduces the deployment complexity.

7. IMPLEMENTATION

The implementation details of our approach are discussed in this section. We use a famous Web application, namely IBuySpy Portal, as a representative benchmark to evaluate the performance of our framework.

7.1 The IBuySpy Benchmark

In order to evaluate the performance of our approach, we use Microsoft.Net IBuySpy Portal as the benchmark. Web sites like this are among the most common Web applications in the Web. IBuySpy Portal is implemented using ASP.Net, and the source code is freely available at [2]. It contains *Web Forms pages* (*aspx* files), *Web Forms user controls* (*ascx* files) and their *code-behind* classes. Similar to the ASP and JSP page, Web Forms pages represent dynamic web pages. The Web Forms user controls represent portions of Web Forms pages. While the aspx and ascx files contain the visual representation, the code-behind classes contain processing logics. When a request arrives, the specified Web Forms page and Web Forms user controls are loaded. The corresponding code-behind objects responsible for generating responses will initiate calls to the request processing. If the processing does not require database interaction, results are returned right away. Otherwise, the processing will generate database queries through a series of specific database access classes (in namespace `PortalCSVS.Components`). But this may cause some response latency to requests. Therefore, if the generated content has already been stored in cache, response latency can be greatly reduced by fetching directly from cache instead of generating them.

Besides caching the entire page, ASP.NET also allows Web Forms user controls to be cached separately. ASP.NET provides duration control for each cached entities (*Web Form* and *Web Form user control*). Duration specifies the life time of a cached page or user control.

7.2 The Fragment-Based Technology

We select one of current fragment-based technology proposed in [16] to use in our prototype. In this case, after all web pages within the site are decomposed into fragments, the number of fragments ranges from one to nine.

When a client requests a web page in the site, web server only generates missing fragments to return them to clients. When the response arrives, client needs to conduct two things. One is to save the missing fragments to local cache, and the other is to fetch cached fragments to make page composition.

We can conveniently write or access local cache using APIs (provided by Microsoft Windows Internet platform SDK). As an explanation, the time cost spent on retrieving a specified fragment from cache is quite small and can be ignored. In our implementation, the cache size is set to 16MB and it is automatically maintained by Microsoft Internet Explorer cache management mechanism.

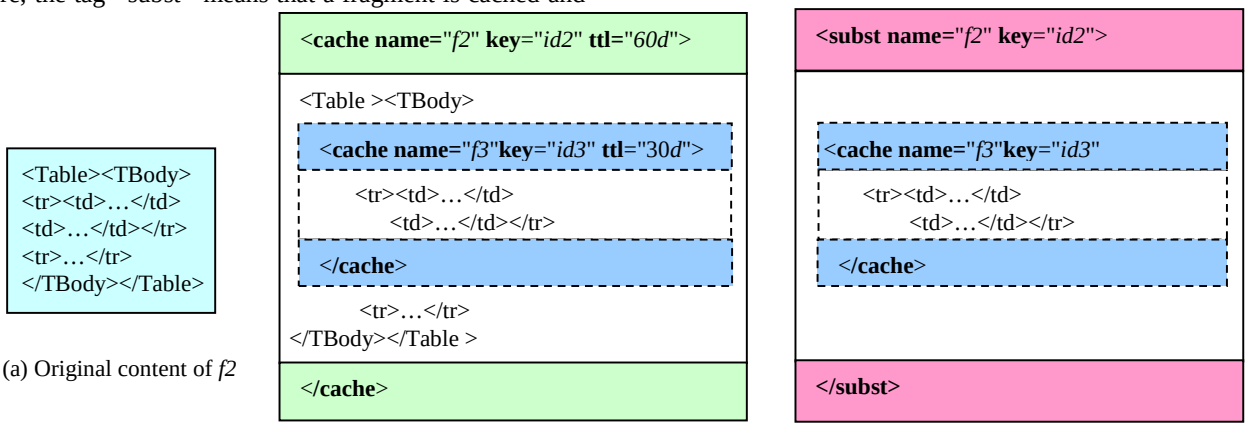
7.2.1 Web Application Modifications

It's a common approach to re-encode web pages with specially defined tags to indicate the cacheable properties of their included cacheable fragments. In our prototype implementation, all dynamic web pages (aspx files) within IBuySpy are to be encoded with fragment definition tags in [16]. In web server, we replace the fragment's placeholder with a special tag that indicates to the client that it is to be expanded with the content of a fragment.

Our implementation currently targets ASP.NET applications and assumes programmers have used ASP.NET Web Forms Page and UserControl class to implement fine-grained

dynamic content caching on server side. To enable this kind of output tagging for such applications, it is sufficient to avoid regeneration of cached content. The modification to the application must satisfy two requirements. The new application should be able to recognize the list of identifiers of cached fragments sent from clients and avoid regeneration of corresponding content. It also needs to insert additional special tags as described to enable the client to do fragment caching and page assembly. Figure 7 provides an example for the generated tagged content of *f2* (Figure 1) in two kinds of different cache hit status. In the figure, the tag <subst> means that a fragment is cached and its content can be fetched from cache by retrieving its identifier indicated by the property *key*. While the tag <cache> means that a fragment is a missing fragment needed to be cached and its lifetime is indicated by its property *ttl*.

As for the detailed algorithm to generate special tags for fragments within ASP.Net dynamic web pages, it can be referred to [16]. As the modification of web applications to be encoded with fragments in ASP.Net web server turns out, the changes are very minor, and even trivial if supports are built into ASP.NET.



(a) Original content of *f2*

(b) Tagged content when *f2* and *f3* are missing

(c) Tagged content when *f2* is missing and *f3* is cached

Figure 7. Tagged content with specified tags which indicate the cacheable properties of fragments.

7.3 Our Prototype

Our framework is to provide a mechanism for assembling a web page and adapting it to present adaptation results to small-screen mobile devices. The implementation must be able to download page fragments, process them to assemble the page, and make page adaptations to display the results. The implementation details of our prototype can be seen from Figure 8. When a mobile client begins to request a web page, cache retrieval module is invoked to examine the cache hit status of fragments within the requested page. According to whether all fragments are validly stored in local cache, two different phases may happen.

1) If fragments are all validly stored in local cache, the request to remote web server will be canceled. Page composition module is called to assemble a whole page by fetching all fragments from local cache. Finally, page adaptation module using our constrained page splitting method is activated to adapt the page for tailored display on small-screen mobile devices.
2) If fragments are partially or completely missing in local cache, fragment download module will be activated to download the missing fragments within the requested page. After the missing fragments are returned from web server, page composition module is called to combine downloaded fragments and cached fragments to assemble a whole page. Then page adaptation module using our constrained page splitting method is activated to adapt page. Finally, the split results of missing fragments will be saved to local cache.

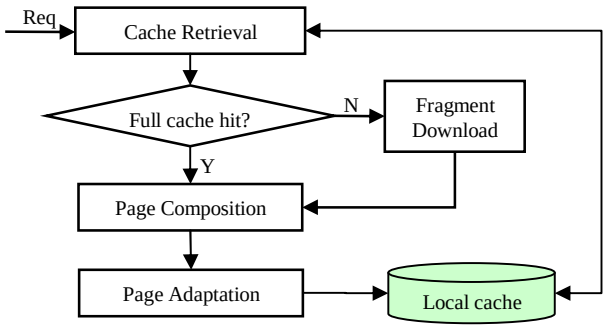


Figure 8. The implementation details.

As aforementioned, fragment download module should be able to download missing fragments instead of whole pages. We achieve this by adding the identifiers of cached fragments to request headers like this:

X-CachedFrags: id1, id2, …, idn

Note that *id1, id2, idn* and so on represent the identifiers of already cached fragments within requested pages. Using these attached headers, the download module requests page fragments using the code snippet below:

```
try {
    httpDoc = HttpOpenRequest("GET", url);
    //Attach identifiers of cached fragments to headers
    HttpAddRequestHeaders(httpDoc, XCachedFrags);
    InternetReadFile(httpDoc);
} catch (Exception e) {<Exception Processing>}
```

When web server receives such a request, it only generates missing fragments other than cached fragments indicated by the request header and returns them to client side.

8. FRAGMENT CACHE HIT RATIO

In traditional caching approaches, cache hit ratio (*CHR*) is defined as the ratio of the number of pages found in cache and the total number of pages requested. It is an important performance criterion for conventional caching algorithms which are based on page granularity. However, our approach is based on fragment granularity. In our unified framework, only missing fragments within a dynamic web page need to be generated, transmitted and split, and processing costs of other cached fragments can be saved. Therefore, *CHR* is no longer a validly efficient criterion to evaluate the performance of our framework.

In this paper, we use another measure, namely Fragment Cache Hit Ratio (*FCHR*) to evaluate our framework. It is defined as the ratio of the number of fragments that hit in cache and the total number of fragments within pages requested. In order to better differentiate the term *FCHR*, we will call traditional *CHR* Page Cache Hit Ratio (*PCHR*). In this case, cache hit for a fragment means that the split result of the fragment has been cached, while cache hit for a page indicates that split results of all fragments within that page have been cached. In our implementation, the method to calculate of *FCHR* and *PCHR* of dynamic web pages is listed in Section 8.1. Their impacts on user-perceived delay will be discussed in Section 8.2.

8.1 *FCHR* and *PCHR*

Suppose there are a collection of n dynamic web pages $P=\{P_i\}(1\le i\le n)$, for example, the dynamic web pages requested within a web site during a period time. First *FCHR* and *PCHR* will be defined for each individual page, and then define the average *FCHR* and the average *PCHR* for the page collection.

Definition 2: For a dynamic web page P_i, its Fragment Cache Hit Ratio (*FCHR*) is defined as:

$$FCHR_i = \frac{N^i_{hf}}{N^i_{hf}+N^i_{mf}} = \frac{N^i_{hf}}{N^i_{visit}\times N_i} \qquad (2)$$

where

N^i_{visit},	number of visits to P_i
N_i,	number of fragments contained in P_i
N^i_{hf},	total number of cached fragments in P_i
N^i_{mf},	total number of missing fragments in P_i

Note that a page may be requested more than once, so N^i_{hf} and N^i_{mf} are respectively the sum of cached fragments and the sum of missing fragments for each page request.

Definition 3: For a dynamic web page P_i, its page cache hit ratio (*PCHR*) is defined as:

$$PCHR_i = \frac{N^i_{hp}}{N^i_{hp}+N^i_{mp}} = \frac{N^i_{hp}}{N^i_{visit}} \qquad (3)$$

where

N^i_{visit},	number of visits to P_i
N^i_{hp},	number of cache hit for P_i
N^i_{mp},	number of cache miss for P_i

Definition 4: For a collection of n web pages $P=\{P_i\}(1\le i\le n)$, the average *FCHR* is defined as:

$$FCHR_{Avg} = \frac{\sum_{i=1}^{n} N^i_{hf}}{\sum_{i=1}^{n} (N^i_{visit}\times N_i)} \qquad (4)$$

The average *PCHR* for P is defined as:

$$PCHR_{Avg} = \frac{\sum_{i=1}^{n} N^i_{hp}}{\sum_{i=1}^{n} N^i_{visit}} \qquad (5)$$

where N^i_{visit}, N^i_{hf}, N_i, N^i_{hp} are defined as same as Definition 1 and Definition 2.
In our later experiments, the calculation of *FCHR* and *PCHR* are based on the definition in this section.

8.2 User-Perceived Delay

User-perceived delay is one of the key factors determining web browsing experience. However, in our framework, user-perceived delay can be reduced by directly fetching split results of cached fragments from local cache. Therefore, user-perceived delay is related to the cache hit status of fragments, i.e. *FCHR*.

Actually network latency will cause additional influence on the performance of our approach too. Especially, when latency becomes relatively great, fragment-based technologies is not beneficial to reducing user-perceived delay any more. In such case, only fully cached pages can avoid the time-consuming latency over the network. Therefore, conventional caching methods based on page granularity are enough to deal with such case. But for current mobile network, network latency is commonly small compared with transmission costs of pages requested. So *FCHR* plays a key role in deciding user-perceived delay of requested pages, as will be shown later in the experimental results.

Eventually, our approach can save the processing costs of processing cached fragments within a page requested. As a consequence, it is evident that user-perceived delay of a requested page can be reduced when it has more included fragments stored in cache. Our experimental results will approve this point later.

9. EXPERIMENTS

In this section, the test bed used in our experiments and its configurations will be introduced first. Then the experimental results for readability and user-perceived delay and their corresponding analysis will be presented. In the end of this section, we also provide strategies to improve *FCHR* and *PCHR* for dynamic web pages.

9.1 The Test Bed

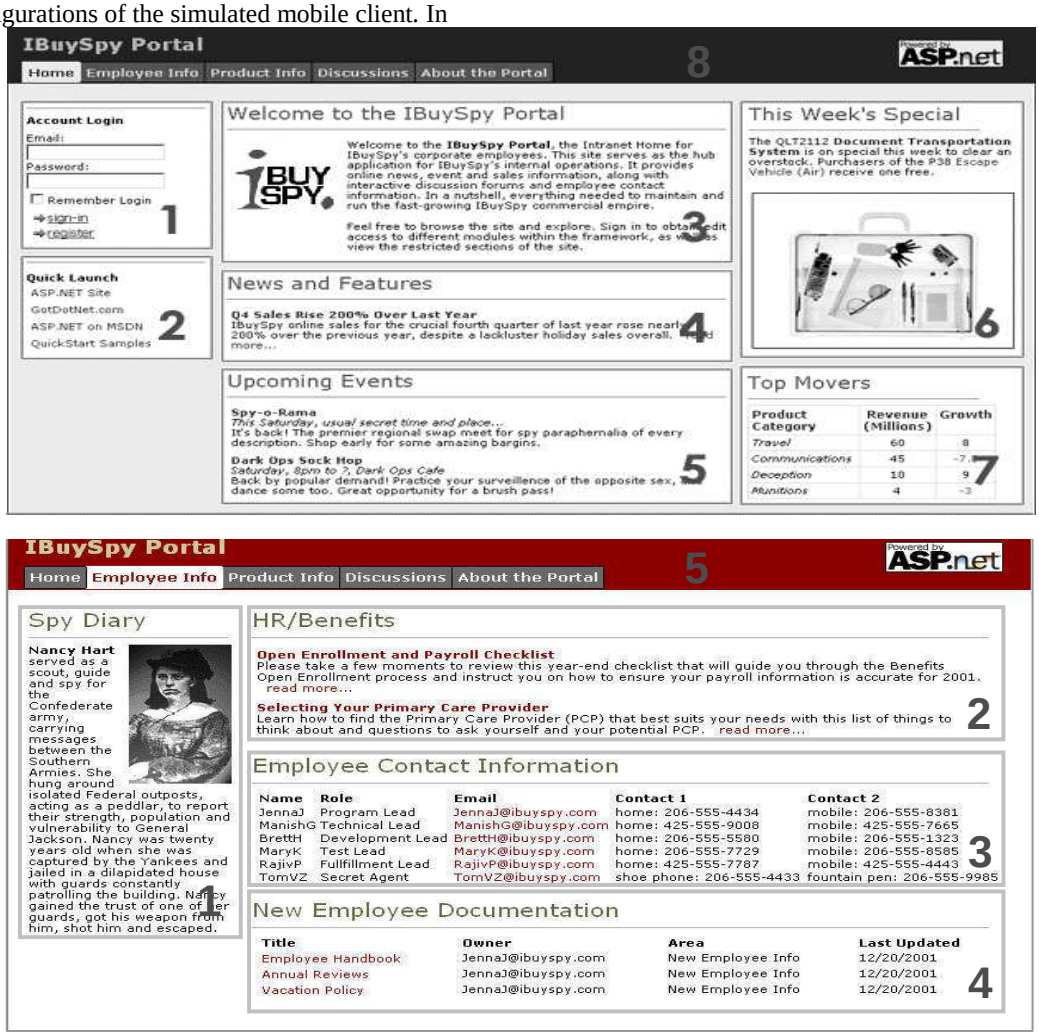
We use two machines in our experiment: a web server and a simulated mobile client. The configurations of the web server are listed in Table 1. Since the underlying support for developing applications on mobile devices is still limited and hard currently, in our experiments we use a low performance desktop computer to simulate the mobile client. Shunra\Cloud [12] is used to emulate the limited wireless network bandwidth and network latency. It resides in the web server and only imposes minor additional loads. Table 2 shows the configurations of the simulated mobile client. In our experiments, we implemented the client web browser capable of our approach, and its size is set to be 240x240 to simulate the small screen of mobile devices.

Table 1. Configurations of the web server.

Hardware	Intel Xeon CPU 3.06GHz, 2GB RAM
Web Server	IIS 6.0 + ASP.NET
DBMS	SQL Server 2000
Operating System	Microsoft Windows Server 2003
Network Emulator	Shunra\Cloud

Table 2. Configurations of the simulated mobile client.

Hardware	Intel Pentium CPU 300MHz, 64 MB RAM
Operating System	Microsoft Windows2000 Professional
Client Browser	Microsoft Web Browser ActiveX



(a) Home page

(b) Employee info page

Figure 9. The split results of two web pages within IBuySpy Website. In (a), the denoted blocks labeled by numbers from 1 to 8 are actually eight fragments contained by the template fragment of the home page. In (b), the denoted blocks labeled by numbers from 1 to 5 are five fragments contained by the template fragment.

9.2 Readability

After being processed by our constrained page splitting method, a dynamic web page is split into a series of tailored blocks which can fit well into the small screen of mobile devices.

In our experiments, we have split a series of requested pages within IBuySpy. According to our observation in the experiments, we found it a common phenomenon that most independent segments (detected in the first step of constrained page splitting method) do not need to be further split. This affirms to us that the time complexity of page adaptation is greatly reduced by using semantic information indicated by fragments.

Figure 9 provides an example of the split results of two web pages generated by our approach. In the figure, the denoted blocks are actually the included fragments within the pages except for their template fragments. Our approach uses the higher level semantic information of fragments instead of only low level HTML tags. As a consequence, the results correspond much better to user's perception.

Figure 10. The distribution of the performance evaluation based on the three categories: perfect, good, and bad.

We conducted a user study to evaluate the readability of our approach. We use web pages within IBuySpy as our test data. Eight participants are asked to give their comments on the adaptation results of our approach. The adaptation result of each web page is presented to testers, and their task is to evaluate our adaptation results. Each evaluation is put into one of the following three categories: perfect, good and bad. Perfect means that the page splitting is perfect without any error. Good means that the page splitting result is correct but there are minor errors and those errors do not affect the overall viewing of results. Bad means that there are errors in page splitting, contradicting user's perception or causing a browsing problem in losing information. Figure 10 shows the distribution of our performance evaluation based on the three categories. About 65% in average assume our results are perfect. Note that more than 90% of our results are either perfect or good. The "bad" category is mainly due to the poor readability of wide blocks (e.g. the banner fragment labeled by 5 and 8 respectively in Figure 9 (a) and (b)) displaying on the narrow width of browser. However, these blocks can't be further split for the indivisibility of their contents. This reminds us of applying other adaptation methods together with ours, such as changing the initial layout of these blocks to be tailored for the small display.

9.3 User-Perceived Delay

9.3.1 User Browsing Pattern

In order to measure the general benefits of user-perceived delay reduction using our approach, we need to get a general browsing pattern of visiting IBuySpy. Thus, using the pattern, we can emulate a general client request to web pages within IBuySpy.

It can be achieved by learning from a set of user logs. Before beginning to record their log, six participants are firstly asked to get familiar with the structure of the IBuySpy site. Then they begin to visit the site according to their own interests. Their visiting experiences are recorded into a log file in this time. Our objective is to acquire a common browsing pattern based on user visiting logs. Table 3 provides the average user browsing distribution, and we think it corresponds to a general user browsing pattern for the web site. Our later requests are based on this browsing distribution.

Table 3. The average browsing distribution.

Activity	Fragments Number	Percentage
Home page	9	20%
Employee Info	6	15%
Product Info	8	20%
Discussion	6	22%
About	9	7%
More Detail	3	11%
Register page	1	5%

9.3.2 Results

As for an example of the time costs for browsing dynamic web pages when no split results have been cached, which is the common case for current page adaptation approaches, take a look at Figure 11. In the figure, we list three pages within IBuySpy as an example to display the user-perceived delay, adaptation cost and transmission cost (under the 25kbps bandwidth). Note that our approach only considers dynamic contents within web pages, and other static objects (e.g. image, etc.) are thought to have already been cached, time costs listed in Figure 11 includes dynamic contents other than static objects. Though the adaptation costs in the table are somehow lower than the transmission costs, we believe that real adaptation cost are much larger because the poor computing power of mobile devices can't be completely emulated by our simulated client. In real mobile environment, we believe it will furthermore support that our approach can save the large adaptation costs.

User-perceived delay of browsing dynamic web pages on mobile devices is mainly caused by the time-consuming

transmission and adaptation, which can be potentially saved. In our experiments, using the time cost of the first visit to web pages as baselines where no fragment is cached, the saved percentage of user-perceived delay is presented to evaluate the performance of our framework. We measure the saved percentage of user-perceived delay and the corresponding *FCHR* under three kinds of network latencies and two types of bandwidths. Note here, *FCHR* and *PCHR* are calculated based on the method described in Section 8.1. The results under three network latencies are shown in Figure 12, where the bandwidth is 25kbps which is typical for a GPRS connection. Each point in the figure stands for an individual request to a page (i.e. N^{i}_{visit}=1 described in Section 8). The data indicates that the saved percentage of user-perceived delay increases with *FCHR*, as expected.

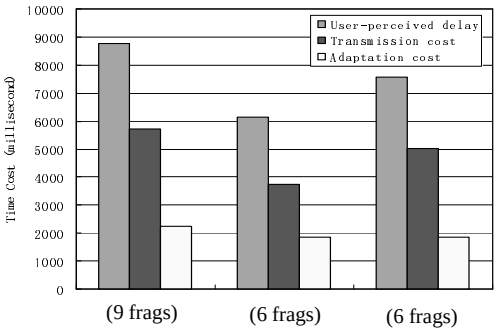


Figure 11. User-perceived delay, transmission cost and adaptation cost for three example pages.

In addition, the saved percentage of user-perceived delay is more dependent on *FCHR* when the network latency is 0, and it becomes deviated from *FCHR* when the network latency gets larger. The figure also shows that there are some little differences between *FCHR* and the saved percentage of user-perceived delay. This is mainly due to the fact that the processing time of each fragment (including generation, transmission and splitting costs) is not fully equal to each other. From Figure 12, we also found that even under a network latency of 2000ms, which is rarely large in current wireless network, *FCHR* still has a non-negligible impact on the user-perceived delay. Therefore, fragment-based technologies will be beneficial in this case. Figure 13 shows the results for the bandwidth which is 56kbps, which is a common case for ordinary modem connection. The data also tells us that *FCHR* can greatly influence the user-perceived delay.

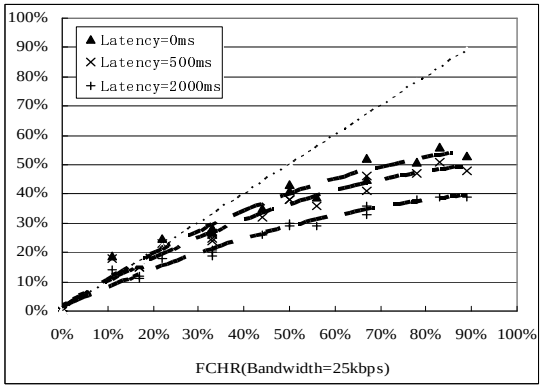


Figure 12. User-perceived delay vs. FCHR (25kbps).

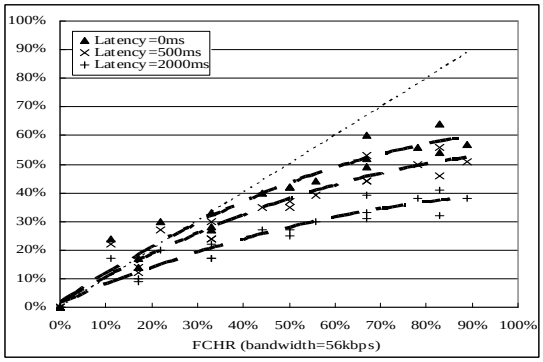


Figure 13. User-perceived delay vs. FCHR (56kbps).

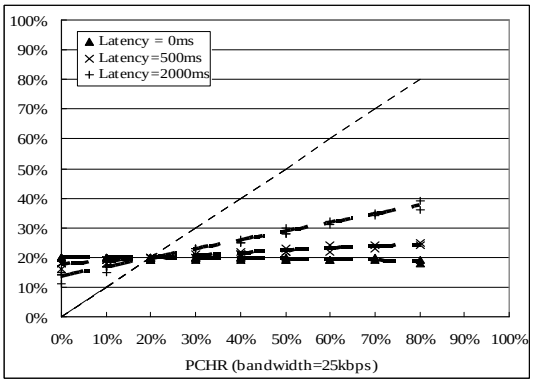


Figure 14. User-perceived delay vs. PCHR (25kbps).

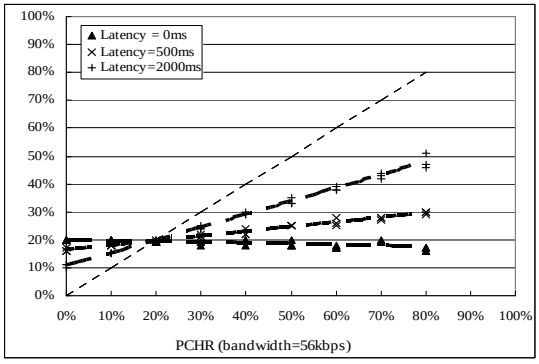


Figure 15. User-perceived delay vs. PCHR (56kbps).

We also compare the saved percentage of user-perceived delay with *PCHR* under the network latencies and bandwidths used above. In the test, we assign the average *FCHR* to be around 0.2 (which can be achieved by setting the cache status of fragments included in pages requested), and measure the saved percentage of user-perceived delay with *PCHR* varying from 0 to 0.8. The results under 25kbps and 56kbps bandwidths are respectively shown in Figure 14 and 15.

The results in both figures indicate that the saved percentage of user-perceived delay is independent of *PCHR*. The saved percentage of user-perceived delay is close to the prearranged average *FCHR* (around 0.2), which further supports that *FCHR* rather than *PCHR* determines user-perceived delay in mobile browsing experience. We can also see from this figure that there is a rough trend for the saved percentage of user-perceived delay to get closer to PCHR with the increase of network latency. This can be extended to that when network latency becomes large enough, fragment caching is no longer beneficial to accelerate user-perceived delay. For instance, mobile network becomes overly jammed or unconnected.

9.4 Improving *FCHR* and *PCHR*

From the experimental results, we can see that *FCHR* is the key factor determining user-perceived delay. However, *PCHR* can better measure user-perceived delay only when the latency of wireless network is large according to our deduction in Section 8.2. Here we present our experiences on how to improve both the *PCHR* and *FCHR* for dynamic web pages.

When wireless network latency becomes relatively large, improving *PCHR* is a critical issue to accelerate response time. Since the lifetime of a whole page depends on the minimal lifetime of fragments within the page, we can try to make a web page composed of fragments with similar lifetimes to prevent the overall lifetime being limited to the minimal ones. Otherwise, expiration of a whole dynamic web page is just limited to its fastest changing fragment inside, regardless of other fragments even with stably long lifetimes.

As stated previously, improving *FCHR* will be more beneficial in case of current mobile network. In our framework, we can successfully save the excessive delay and processing load for cached fragments. To our knowledge, we can carry out two methods to improve *FCHR*. First, we can try to share fragments across different dynamic web pages. This can increase the cache usage for these shared fragments. Second, we should try to make a fragment be composed of contents with similar lifetimes. Otherwise, its lifetime will be limited to the fastest changing content inside.

10. CONCLUDING REMARKS

In this paper, we proposed a unified framework which considers improving both readability and user-perceived delay for browsing dynamic web pages on mobile devices. In our framework, constrained page splitting was presented to split a dynamic web page to improve the readability. Then the split results were saved to cache in fragment granularity. We provided implementation details of our framework. The performance of our approach was measured by a representative Web benchmark. Experimental results point that *FCHR* plays a key role rather than the traditional criterion *PCHR*. Finally, we provided strategies on how to improve *FCHR* and *PCHR* for dynamic web pages.

We are planning to apply our framework to a wider variety of existing fragment-based technologies, which will make our approach more compatible with current caching systems. Another promising direction is to extend our framework to cooperate with existing server or edge caching mechanisms in order to improve the overall performance. With the satisfactory simulation results, we are also considering building our work into web browser in mobile devices, such as Pocket IE. We will continue to investigate these directions in our future work.

11. REFERENCES

[1] Amiri, K., Park, S., Tewari, R. and Padmanabhan, S. DBProxy: A Self-Managing Edge-of-Network Data Cache. Proc. of the IEEE International Conference on Data Engineering (ICDE), Bangalore, India, March 5-8, 2003.

[2] ASP.NET Web: The Official Microsoft ASP.NET Site for IBuySpy Portal. http://www.asp.net/Default.aspx?tabindex=5&tabid=42

[3] Buyukkokten, O., Garcia-Molina, H. and Paepcke, A. Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices. Proc. of WWW-10, May 1-5, 2001, Hong Kong.

[4] Chen, J.L., Zhou, B.Y., Shi, J., Zhang, H.J. and Wu, Q.F. Function-based Object Model Towards Website Adaptation. Proc. of WWW-10, May 1-5, 2001, Hong Kong.

[5] Chen, Y., Ma, W.Y and Zhang, H.J. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. Proc. of WWW-12, May 20-24, 2003, Budapest, Hungary.

[6] Edge Side Includes. http://www.esi.org

[7] Gu, X.D., Chen, J.L., Ma, W.Y., Chen, G.L. Visual Based Content Understanding towards Web Adaptation. 2nd Intl. Conf. on Adaptive Hypermedia and Adaptive Web Based Systems. Malaga, Spain, May 2002`.

[8] Hori, M., Kondoh, G., Ono, K., Hirose, S. and Singhal, S. Annotation-Based Web Content Transcoding. Proc. of WWW-9, Amsterdam, Holland, May 2000.

[9] Li, W.S., Hsuing, W.P., Kalashnikov, D.V., Sion, R., Po, O., Agrawal, D. and Candan, K.S. Issues and Evaluations of Caching Solutions for Web Application Acceleration. In: The 28th Int. Conf. on Very Large Data Bases, Hong Kong, China, August 20-23, 2002.

[10] Rabinovich, M., Xiao, Z. and Douglis, F. Moving Edge-Side Includes to the Real Edge: the Clients. 4th USENIX Symposium on Internet Technologies and Systems. 2003. March 26-28, 2003, Seattle, U.S.A.

[11] Milic-Frayling, N. and Sommerer, R. SmartView: Flexible Viewing of Web Page Contents. Poster paper at Proc. of WWW-11, Hawaii, 2002.

[12] Shunra\Cloud. http://www.shunra.com/cloud.htm

[13] Stuary, G., Rag, T., Sreedhar, K. ATTENUATOR: Towards Preserving Originally Appearance of Large Documents When Rendered on Small Screen. ICME2003, July 6-9, 2003, Baltimore, U.S.A.

[14] Yagoub, K., Florescu, D., Valduriez, P. and Issarny, V. Caching Strategies for Data-Intensive Web Sites. Proc. of the Int. Conf. on Very Large Data Bases, Cairo, Egypt, 10-14 September, 2000.

[15] Yuan, C., Chen, Y. and Zhang, Z. Evaluation of Edge Caching/Offloading for Dynamic Content Delivery. Proc. of WWW-12, Budapest, Hungary, 2003.

[16] Yuan C., Hua, Z.G. and Zhang, Z. Proxy+: Simple Proxy Augmentation for Dynamic Content Processing, IWCW2003, NY, USA, September 29-October 1, 2003.