Journal of Web Engineering, Vol. 4, No. 1 (2005) 079–099
© Rinton Press

ACCELERATING DYNAMIC WEB CONTENT DELIVERY USING KEYWORD-BASED FRAGMENT DETECTION[a]

DANIEL BRODIE[b], AMRISH GUPTA, WEISONG SHI
Department of Computer Science, Wayne State University
Detroit, Michigan, 48202, USA
{*daniel,amrish,weisong*}*@wayne.edu*

Received August 8, 2004
Revised December 14, 2004

Recent advances in Web engineering have enabled the rapid growth of dynamic Web services such as Web-based email, online banking, online shopping and entertainment. We envision that finding an effective way to deliver these dynamic Web services and understanding the relationship between Web application design and delivery are two important Web engineering issues, and have not been seriously considered in the community. In this paper, we intend to tackle the first problem and pave the way for solving the second problem in the future .

To efficiently serve this trend, several server-side and cache-side fragment-based techniques, which exploit reuse of Web pages at the sub-document (also known as fragment) level, have been proposed. Most of these techniques do not focus on the creation of the fragmented content from existing dynamic content. Also, existing caching techniques do not support fragment movement across the document, a common behavior in dynamic Web content.

This paper presents two proposals that we have suggested to solve these problems. The first, DyCA, a dynamic content adapter, takes original dynamic Web content and converts it to fragment-enabled content. Thus the dynamic parts of the document are separated into separate fragments from the static template of the document. This is dependent on our proposed keyword-based fragment detection approach that uses predefined keywords to find these fragments and to split them out of the core document. Our second proposal, an augmentation to the ESI standard, allows splitting the information of the position of each fragment in the template from the template data itself by using a mapping table. Using this, a fragment enabled cache can have a more fine grained level of identifying fragments independent of their location on the template, which enables it to take into account fragment behaviors such as fragment movement. We used the content taken from three real Web sites to achieve a detailed performance evaluation of our proposals. Our results show that our keyword-based approach for fragment detection and extraction provides us with cacheable fragments that, when combined with our proposed mapping table augmentation, can provide significant advantages for fragment-based Web caching of existing dynamic Web content.

Keywords: Object Characteristics, Dynamic Web Content, Dynamic Content Adapter, Fragment-based Caching

[a]This work was supported in part by the University Faculty Research grant of Wayne State University.
[b]Daniel Brodie was supported by the University Undergraduate Research program of Wayne State University.

1 Introduction

Recent advances in Web engineering has enabled the rapid growth of dynamic Web-based services, which dynamically generate their Web content to serve user's requests in a more timely and/or customizable way, exemplified by services such as `My Yahoo!`, `myciti.com` [7, 9, 17, 26, 45, 50]. On the one hand, these dynamic Web services improve the usability of Web applications in a tremendous way. On the other hand, the prevalence of these type of services and applications introduces several Web engineering challenges, including Web application design and implementation, deployment and maintenance, and delivery across the wide-area network. Although some of these issues have attracted a lot of attention in the Web engineering community in recent years [6, 11, 41], we argue that finding an effective way to deliver these fast growing dynamic Web services and understanding the relationship between Web application design and delivery have not been seriously considered in the community. In this paper, we take the initial step to tackle the first issue by examining examine the efficient way for dynamic Web service delivery on the Internet in the context of Web caching.

Web caching and content distribution networks (CDN) are popular solutions for improving Web access latency and have the effect of moving content closer to the client. However, these solutions typically do not work well with dynamically generated and personalized content [25]. Researchers have recently proposed several server-side and cache-side mechanisms to improve the generation and serving of dynamic Web content. Server-side techniques, exemplified by techniques such as delta encoding [29], data update propagation [12], fragment-based page generation [13, 16], reduce the load on the server by allowing reuse of previously generated content to serve new requests. Cache-side techniques, exemplified by systems such as Active Cache [10], Gemini [30], CONCA [39], and the content assembly technique proposed by Wills *et al.* [44], attempt to reduce the latency of dynamic content delivery by moving some functionality to the edge of network. Similar trends are also visible in commercial caching and edge server products, most notably IBM's WebSphere [21] and Akamai's Edgesuite [1].

Despite their difference in focus, both server-side and cache-side approaches share the same rationale, specifically that it is possible to view the document in terms of a quasi-static *template* (expressed using formatting languages such as XSL-FO [46] or, what is currently becoming the *de facto* standard, edge-side include (ESI) [42]), which is filled out with multiple, individually cacheable and/or uncacheable *objects*[c]. This object composition assumption enables surrogates and downstream proxy caches to reuse templates and cached objects to efficiently serve subsequent requests and additionally reduce server load, bandwidth requirements, and user-perceived latencies by allowing only the modified or unavailable objects to be fetched.

From the perspective of Web engineering, the above proposed techniques can be broadly classified into two school of thoughts, *content caching* and *function caching*, which could be used for either server side proxies or client side proxies. In the content caching, a dynamic Web page is cached as different fragments/channels. These fragments/channels are maintained as separate objects in the edge server's cache and are dynamically assembled into Web pages using XML/JavaScript type languages by fetching only non cacheable or expired fragments/channels from the origin server in response to user requests. The origin server supports this assembly and the exchange of information between origin server and proxy is XML type

[c]The terms *objects* and *fragments* will used interchangeably in this paper.

data. The representatives of this approach include Akamai [1], CONCA [39], and Client Side Include (CSI) [34]. All of them are built upon the edge-side include (ESI) technology [42].

In the function caching, part of a Web application is replicated and cached in edge servers (also known as proxy caches) along with its associated applications so that part of the Web application runs at the edge servers instead of the origin server. The exchange of information between the origin server and proxy is the application itself. The three-tier architecture of most Web applications fits this trend very well, by migrating the presentation tier to the edge server, and keeping the business logic and database access in the original server. This has been exploited in the recent study in [19, 47]. Examples of this method are vMatrix [4], IBM Websphere [21], Active Cache [10], Gemini [30], Proxy+ [48], and SEE [27]. Function caching makes content delivery closely coupled with Web application deployment and configuration, and is a very interesting Web engineering topic. However, in this paper, we will study the content caching technique, and leave the function caching as our future work.

Although the above techniques appear promising, there are a number of issues that are not addressed in these current infrastructures. Even though there might be techniques used by certain companies [1], due to their closed nature we cannot check them, and so to the best of our knowledge, there is no open and free method of separating objects from existing dynamic document, except from our existing work on DYCE [37]. Also, current technologies for supporting dynamic objects do not differentiate between the location of the objects in the document, and object itself. This makes it difficult to efficiently implement the situations where the object can move between different places in the document without changing data, which is common in certain news Web sites [40]. More detailed explain is in Section 3.

This paper describes our efforts on addressing these shortcomings. We are proposing two methods that should solve these shortcomings. The first is an augmentation to the ESI standard, the most used method for specifying the format of the templates, to allow the fragment locations to be specified in a mapping table that is sent with the template. This allows the objects to move across the document without needing to re-serve the template. Our second proposal, DyCA, a Dynamic Content Adapter, is a two part model for creating object-based content from original dynamic content. The first part extracts the objects from the original content, giving us the needed separation between template, objects, and object location by using our mapping table approach. The second part of DyCA delivers the content to a fragment-enabled client, like a caching proxy server. A Python-based fragment-enabled caching proxy, named CONCA-Lite was developed to allow the testing of the object extraction, and content delivery modules of DyCA.

Our method for creating the dynamic content from the original Web content is based on a simple and effective keyword based object extraction technique to find dynamic objects inside a static Web page. The dynamic content can then be served by our DyCA server, which can serve fragments from the document as needed, enabling a client to support template and object caching. Our proposed ESI-extended format allows for caching of both the objects and the template and allows for object movement. This type of concept, where the object in the template maps, based on a mapping table, is, to our knowledge, introduced here for the first time. By having a fully functional fragment-enabled content server and client, and by testing on real world data, we have gotten accurate results, beyond regular experiments done in the field. These results have shown that our proposed method for fragment extraction based on

the keywords in the document can allow us to cache existing non-fragmented content and achieve significant performance improvements by utilizing our proposed augmentation for a mapping-table based template.

The rest of the paper is organized as follows. Section 2 describes the related background for this field. Section 3 addresses problems with the current infrastructures. Section 4 shows the design and implementation of the system. Section 5 presents the evaluation and results of the testing of our architecture. Section 6 concludes the paper and discusses our planned future work.

2 Background

The focus of this paper is studying the enabling technologies to accelerate the dynamic Web content delivery, which are built upon several existing technologies. In this section, we in turn present four important concepts that related to the rest of the presentation, including *dynamic Web content and web caching*, *fragment-based caching*, *edge side include markup language*, and *CONCA architecture*.

2.1 Dynamic Web Content and Web Caching

Dynamic Web content can be broadly classified into two types. The first type which we henceforth call just *dynamic pages* are those that are generated without taking sessions of the user into account. This type does not need to know who has accessed the page and for every user the dynamic page generated is the same at any instant of time. The second type, which we call *personalized dynamic pages* are generated when the user accesses them through a secured system. In this case the dynamic page generated is tailor made for each user. From Web caching view of point, both dynamic pages and personalized pages are considered uncacheable because of its dynamic nature as explained in the next paragraph. In this paper, we will treat both dynamic and personalized pages as *dynamic Web content*.

Both Web caching and content distribution network attempt to improve Web application performance in three ways: (1) to reduce the user-perceived latency, (2) to lower the network bandwidth usage on the network, and (3) to lighten loads on origin servers. These potential benefits are promising, but to what extent they can be achieved is tightly dependent on the lifetime (or time to live) of Web content. If a Web content is dynamically generated for every request (i.e., two calls to the same script with the same input parameters might not produce the same output), it will not be cached in current Web caches and content distributed networks (CDNs). Furthermore, in order to suit diverse interests of multiple customers, many Web sites support personalized web pages that are generated dynamically for different people by using a `cookie` HTTP header [17]. Unfortunately, those contents are being considered as uncacheable in current technologies. In the last five years, we have witnessed the fast growth of dynamic and personalized Web content [7, 50], observing that both Web caching and CDN typically do not work well with those content [17, 25, 45]. Therefore, a totally new school of thoughts is needed in order to improve the performance of Web applications delivery.

2.2 Fragment Based Caching

In order to accelerating the delivery of dynamic Web services, several researchers have examined the characteristics of dynamic and personalized Web content, and concluded that

the majority of these contents is "reusable" [38, 43] after splitting the whole content into small fragments (also known as objects in [38].). Next, we will present the basic idea of fragment-based caching.

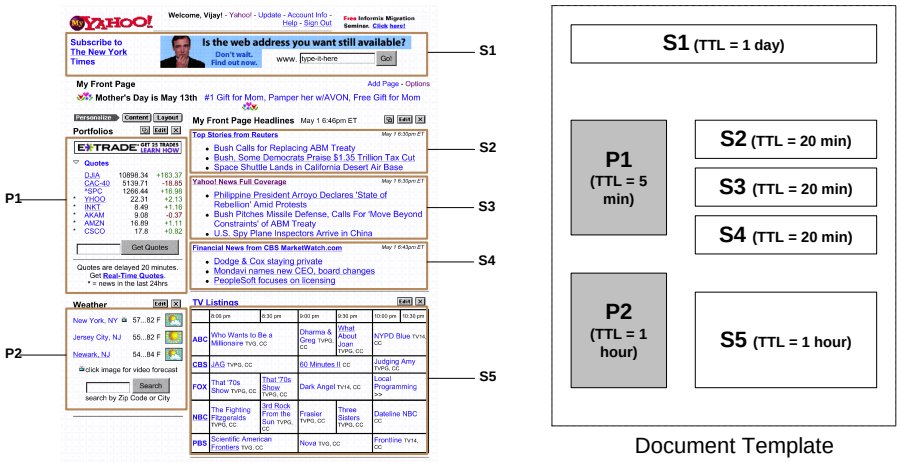

Fig. 1. Dynamic content can be viewed in terms of a quasi-static document template and individual objects, which exhibit different sharing, cacheability, and freshness time characteristics.

One fundamental block in caching dynamic content is allowing to split up a document into different static and dynamic parts. By doing this, parts of the document, called *fragments*, can be treated separately rather then treating the document as a whole. Thus each fragment can have its own behavior allowing more fine-grained control over caching behavior and data sharing of the different segments of a document. Certain fragments of a document can be shared between different clients whereas some clients want different information in other fragments. Also, some parts of a document change more frequently than others, while other parts are completely static. By treating the document as a whole rather then separating it into fragments a page cannot be partially shared between users nor can it be partially cached. Rather, when a little part of the document changes the whole document needs re-fetching, and if parts of the document can't be shared, then the document can't be shared at all. For example, consider a popular customizable Web site with content that gets continuously updated with information such as news and weather. Figure 1 shows the snapshot of a personalized `my.yahoo.com` page, which fits in such an example, and what the corresponding document template and component objects might look like. `S*` and `P*` represent objects that are shared and private respectively, and `TTL` captures the length of time this object remains valid. In the figure, the blocks (S* or P*) on document template (right side) map exactly the corresponding blocks of the original content (left side). The contents of fragments on the page can change as new stories develop or as the weather changes, leaving other fragments unchanged and with no need to re-fetch the data. Users that are viewing sports stories can share those fragments, and if some of those users are also viewing the economic section, then that can be shared by all the people viewing that fragment as well. There currently are a number of ways to split a document into fragments and reconstruct it. Different approaches do reconstruction in different parts of the network, including the originating server, the cache proxy, and the client. Each one of these methods has a different way of dealing with caching

and the fragments.

2.3 Edge-Side Includes

The ESI [42], Edge-Side Include, has currently become the de facto standard in specifying the format for templates in fragment based documents. It uses a simple XML-based markup format that specifies content fragments for inclusion and dynamic assembly in a base template. These ESI specific tags are provided as extensions to the traditional HTML format allowing for minimal format change to the original document format. It separates the document in such a way that allows for the server or proxy to manage the objects as separate entities. This allows for different levels of cacheability for each fragment, and for large amount of dynamic content to be cached. Figure 2 shows an example of ESI template within a table structure in HTML document. The `<!--esi` and `esi:remove` tags allow templates to be handled by non-ESI downstream proxy caches and Web browsers. If the template is received by non ESI-enabled browser or proxy cache, the browser or proxy cache will assume the line 4 signifies the beginning of HTML comments and will ignore lines 4-13, and instead display `non-esi.html` by ignoring the `esi:remove` tags. On the contrary, an ESI-enabled proxy cache or browser will remove `<!--esi` tags from the final page, and treat the content within the `esi:remove` block as comments. So, lines 5-12 will be processed further. The basic structure of an ESI fragment `esi:try` tag, which consists of two parts, `esi:attempt` and `esi:except`. In our example, if the attempt succeeds, it will display the fragment `o1.html`, otherwise, the fragment `default.html` will be downloaded. From this example, we can see that ESI language can be used to describe finer-granularity objects than are currently identified using hyperlinks in top-level documents (e.g., the HTML HREF tag). The latter are already amenable to sharing using conventional caching architectures. More appropriate for our purpose are sub-document entities such as HTML frames, tables, paragraphs, etc.

```
1. <table>
2. <tr>
3. <td colspan="2">
4. <!-- esi
5.       <esi:try>
6.       <esi:attempt>
7.               <esi:include src="/o1.html" onerror="continue"/>
8.       </esi:attempt>
9.       <esi:except>
10.              <esi:include src=/default.html />
11.   </esi:except>
12.   </esi:try>
13. -->
14. <esi:remove>
15.      <A href=/non-esi.html> non-ESI version </A> of this page
16. </esi:remove>
17. </td>
18. </tr>
19. </table>
```

Fig. 2. An example of ESI usage.

ESI also includes a very complete framework for conditional getting of fragments, cookie support, and error control. Every `esi:include` tag references a specific URI with a TTL (time to live) value, all of this is included in the template file which gives the layout and aesthetic look to the document. Thus, all the information regarding the fragments and the template is actually sent in the template itself. For a client to support the ESI framework all that it needs to do is to implement support for parsing and acting based on the template format. Thus, ESI's simplicity is a very strong point.

2.4 CONCA Architecture

The work presented in this paper is part of the umbrella project CONCA proposed in [39]. CONCA is a proposed edge architecture for the efficient caching and delivery of dynamic and personalized content to users who access this content by using diverse devices and connection technologies [39]. CONCA attempts to exploit reuse at the granularity of individual objects making up a document, improving user experience by combining caching, prefetching, and transcoding operations as appropriate.

To achieve its goals, CONCA relies on additional information from both servers and users. All content supplied by servers in CONCA architecture is assumed to be associated with a "document template" which can be expressed by formatting languages such as XSL-FO [46] or Edge Side Includes (ESI) [42]. Given this information, CONCA node can efficiently cache dynamic and personalized content by storing quasi-static document templates and using the sharable objects among multiple users. Moreover, based on the preference information provided by users, a CONCA cache node delivers the same content to different users in a variety of formats using transcoding and reformatting. The proposed fragment detection approach and dynamic Web content adapter will be used to evaluate the performance of CONCA design in the future.

3 Existing Problems of ESI

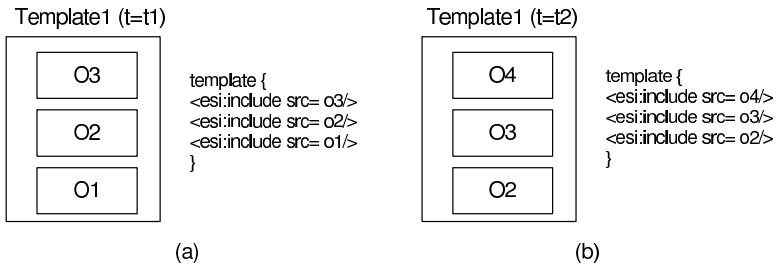

Fig. 3. An example of template caching problem: (a) template at time t1 and (b) template at time t2.

An important factor of the efficiency of fragment based documents is the method used to update it. One of the popular uses in fragment documents is for object movement. This type of behavior, as can be seen in Figure 3, is represented when one or more fragments from a dynamic document move between the different available positions on the template. It can be seen from the figure that object `O3` has moved from the top at time `t1` to the middle of the Web page at the time `t2` (`t2` > `t1`). A good example is a news Web site where old stories (represented as fragments) move down the document and new ones are added from the top.

If we use ESI to construct our document then there are two possible ways to update the document so that the objects can move across the document. Since in ESI the document is split up into two parts, the template, and the fragments, then one possibility will be to update the fragments where the object moved to where it was moved from. Another possibility is to update the template such that the updated template has the URL of the new locations in the template pointing to the correct fragment.

3.1 ESI - Static Template

One obvious solution to fragment movement, which we will call *static template*, is to update the fragments themselves, leaving the template completely static. Using our news Web site example from earlier, the news template points to a series of objects, and when a new story gets added to the page it pushes the old one out of the page. All the objects in the page need to expire, be invalidated, and be re-retrieved as the fragments lower down on the page. Considering that a document, like a news Web site, that has most of its objects moving around the Web page, this means invalidating most of the objects on the page and fetching them from the server on a regular basis. The biggest problem with this method is that most of these fragments are already present on the client just in a different location, and re-fetching them in a different place means a lot of wasted data transfer that could otherwise be cached.

3.2 ESI - Static Objects

A different approach to solve this problem, which we will call *static objects*, is to invalidate the template and re-fetch it for spatial changes, leaving the objects static for such a change. The fragments will still contain only dynamic data and would need to be re-fetched for a data change. The new template will have the object in the first position pointing to the new objects, and all the other positions pointing to the moved objects. This seems to solve the problem since the objects unmodified do not need to be re-fetched. Supposedly, this saves a lot of data transfer by transferring only the new objects and not requiring all the objects to be re-fetched as in our earlier example. Since the only thing that has actually changed in the template is the URL of a few of the fragments, this means that most of the transferred template, which would still contain only static data, is already on the client. Considering that the template can be very large, as we show later in Section 5, doing this on a regular basis, and transferring all this data that could otherwise be cached, this is not a very efficient solution as it might seem at first.

3.3 Mapping Table

There is no way to solve the problem of object movement in the current ESI infrastructure. Either you will be invalidating many objects that really are valid, or you will be invalidating a template which has very little data actually modified. That is why a proposed solution needs to make additions to the ESI infrastructure. These additions, a mapping table that gets sent along with the template and an addition to the template format to allow inclusion of objects from the mapping table, allow for a new method of fragment-based caching, which we will denote as *mapping table*, which does not require either the objects or the templates to be invalidated for spatial changes. The mapping table is just a list of the object ids and their corresponding URLs in a parse-able file. Thus, when a client retrieves a template, the client will also cache the mapping table that was sent with the template. When this client needs to

fetch a fragment from the template, the identifier in the template is looked up in the mapping table, and then the fragment is fetched from the appropriate URL. The mapping table is relatively a small amount of data compared to the template and object sizes. When an object needs to be moved across a document, from one locations in the template into another, the only thing that needs to be updated is the mapping table. Thus, both the template and the objects continue to be cached and treated efficiently, while still allowing for object movement.

It is worth noting that the key to the success of the mapping table technology is the assurance of the small overhead (storage size) resulting from the mapping table itself. From our design, it is easy to see that for each Web content the size of mapping table is linear proportional to the number of fragments extracted from a web content. This number usually is less than twenty for most of dynamic Web contents, as validated in our experimental results in Section 5. In the real implementation, the size of mapping table in a proxy cache is tunable by controlling the number of fragments.

4 DyCA: Dynamic Content Adapter

Although fragment-based caching is a very promising technique to improve the performance of Web content delivery, lacking of fragment-enabled Web server and/or application server that can automatically provide semantic-related fragments is still a big obstacle to the wide deployment of this technique. The major obstacle is lacking of effective Web engineering mechanism to exposure templates and meaningful fragments of Web content to the outside, such as downstream proxy caches and surrogates. As a matter of fact, if one has access to the Web server and/or application server of content providers, it is a relatively straightforward exercise to instrument them to serving templates and meaningful fragments, because that most Web applications already have this piece of information as exploited in [13] for server-side fragment caching. Thus, we envision that the future design of Web applications should consider appropriate mechanisms to exposure their content (including the code for generating the content) in fine granularity in addition to the whole content itself. However, it is still a long way to make this become a reality, including both technical and social factors. Next, we propose a short-term solution to this problem, which can be incrementally deployed on the current Web framework.

To address this problem, we propose to build DyCA, a dynamic content adapter, that transparently takes as input original dynamic Web content and produces as output fragment-enabled content. Not only will DyCA be used in the performance evaluation for downstream caching policies, but DyCA will serve as an efficient front-end for existing dynamic content Websites to incrementally deploy the fragment-based caching technology.

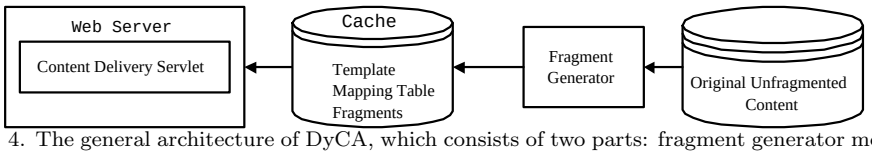

Fig. 4. The general architecture of DyCA, which consists of two parts: fragment generator module and content delivery module.

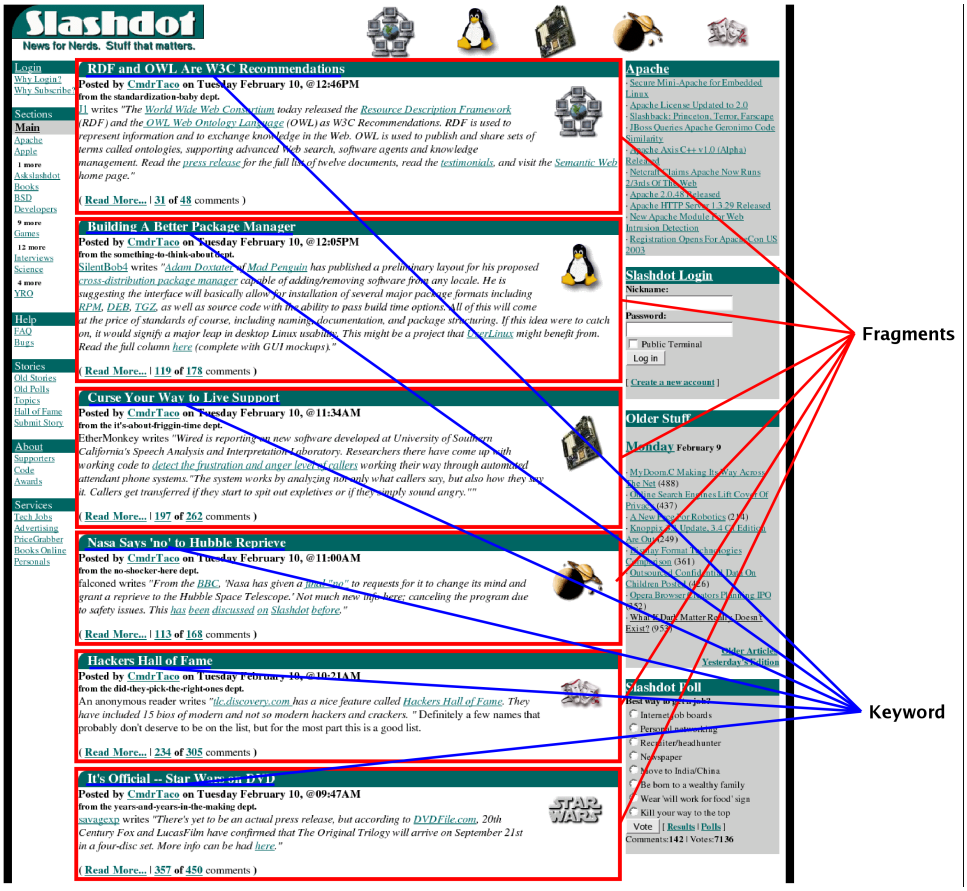

Fig. 5. Fragments in a dynamic Web page can easily be detected based on font size, or other text attributes.

4.1 Design

As mentioned earlier, DyCA, our proposed **Dy**namic **C**ontent **A**dapter, has been designed to augment the existing servers to serve dynamic content efficiently. In the client-proxy-server model it sits on the front-end of the server and can then take existing Web content that is requested from the server and process it to serve the dynamic content instead. So when the original content changes, DyCA will regenerate its fragment-enabled content. This allows it to be deployed anywhere from the same location as the actual Web server or to the ISP level. As shown in Figure 4, DyCA is actually split up into two separate but very important parts, the fragment generator module (the second rectangular box), and the content delivery module (the first rectangular box). The generator module deals with taking the existing dynamic content and converting it into fragmented content. The delivery module can then take the dynamic Web pages generated by the earlier module, and serve them to the client appropriately. These two modules together let us take an existing static Web site and easily change it into a fragment-based dynamic Web site, that can be cached properly. The arrow lines in the figure show the flow of content processing in DyCA. Note that the cache component located between the fragment generator and content delivery is used to implement server-side

fragment caching, as exploited in several previous work [12, 49].

4.1.1 Fragment Generation Module: Keyword-based

The content generation module generates the objects and template from the original dynamic Web content. The most challenging part of our approach is how do we identify the logical objects from the html page. Ideally, we should take into consideration document semantics such as the relationship of objects with each other. In the absence of such information, we have to use heuristic means. We propose to use a keyword based approach to split up the dynamic content into fragments. The keyword based approach works by building an XML of the Web page and searching this XML tree for specific tags that can signify a different object. These tags are set separately for different Web site based on the structure of the HTML and the content. By looking at popular Web sites, such as the personalized `my.yahoo.com` page shown earlier in figure 1, it is fairly simple to see the implicit fragments contained in the document. Fragments can be easily distinguished based on certain differences such as a different font or a table tag, and based on certain predefined keywords, such as the TV Listings headlines, or the Weather headlines. Once the tags in the XML tree are identified, the children XML tags and the rest of the XML content contained in the identified tags is extracted to create the objects. A special include tag, that has a special object id for each object, will be placed in the position where the object was extracted from the main document. Figure 5 illustrates another example of the concept of fragments and keywords. By comparing Figure 1 and Figure 5, we can see that each website has several keywords, which can be pre-determined, that specific to their Web application. For example, in the default New York Times Web page, there are 14 categories of news and 4 headline news, e.g., `BUSINESS`, `INTERNATIONAL`, `TECHNOLOGY`, etc. Intuitively, these small pieces of information are good candidates for the object, and it is very useful to separate them out as an independent object in our analysis. Keyword-based fragment detection technique groups fragments based on the document semantics as much as possible, which is more realistic than the two previous approaches [38]. Note that the applicability of our proposed approach is based on the assumption that the content below a keyword in the constructed XML tree equals to the semantic fragment represented by the keyword. We have been unable to find a Web site whose pages are fully dynamically generated, where this assumption is not satisfied.

Additional information for each object, such as the TTL of the object, is calculated by looking at a long term overview of the Web site. Numerous instances of the Web page are collected over a regular period of time. Each instance is parsed and separated into the different objects, template, and mapping table by the content generation module. By comparing the objects and their position over the period of time, an accurate dynamic behavior can be seen that allows the correct generation of the mapping table to be most efficient with regards to object movement. It also allows the TTL for each fragment location in a document to be generated. Since the location in the TTL of a location in the template remains mostly the same, this TTL is then reused later as the TTL for each fragment location in the document. In our news Web site example, the sidebar listing all the news from yesterday will always have a TTL of 24 hours.

4.1.2 Content Delivery Module

The content delivery module is responsible for serving the template, objects, and mapping table to the fragment-enabled proxy cache. The content delivery module uses the data created by the content generation module. The content delivery module needs to implement the extensions to the protocols in order to send the data created by the content generation module in an appropriate way. It needs to add information regarding the mapping table, and so notify its client, the fragment-enabled proxy cache, when a request is dynamic and has a template or when it is static. By sending the mapping table to the client, the client can then get the static URLs of the objects to be able to access them from the template. The content delivery module also needs to support the client updating only its mapping table, so that bandwidth would not need to be wasted with already cached objects, or templates. This module can be backwards compatible with existing technologies and can support serving to both client level caching [34], and proxy level caching [1, 39].

4.2 Implementation

The dynamic content generator is a program that parses existing Web sites and outputs a dynamic, fragment based, cacheable, Web page. Three Web sites, New York Times [32], India Times [22], and Slashdot.com [40], were chosen due to their dynamic nature, and since none of them supported any form of cacheable dynamic data. The Web sites were constantly monitored for changes during an extensive period covering two weeks. This data was then passed through the keyword based object extraction. Each object was extracted from each instance of the Web site by finding appropriate, predefined tags in the document. Once each object was extracted and the ESI-based template was constructed, the resulting fragmented documents were compared across their time element to calculate the TTL and the mapping table. Object movement across the document is taken into account and allows for the object to not be replaced too soon, and remain in the mapping table. An object was considered expired once it wasn't in any part of the document. Once the template, objects, and mapping

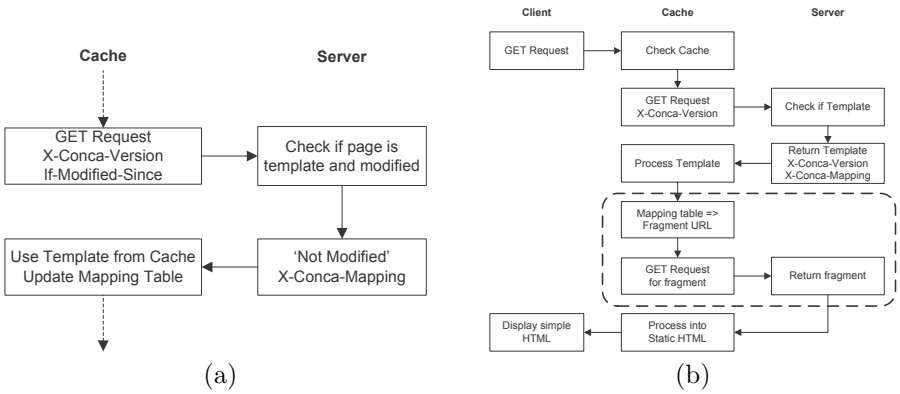

Fig. 6. (a) Process of initial retrieval (b) Process of cached retrieval.

table are in place from the generator module, the dynamic content delivery module just needs to serve them. Implemented as a server extension using servlets in Java and sitting on top of popular Web servers such as the Apache Web Server [2] or the Jigsaw Web Server [23], this module appears as a traditional Web server to regular clients, but provides the dynamic

content ability to able clients. This module uses information from the generator module to build up information such as which pages are templates, the TTL for certain objects, and the mapping table. Figure 6 shows the process of events when a client request for a Web page. When a CONCA-enabled request from a proxy for a document is made, if the document has fragments then the mapping table is looked up and added as an `X-CONCA-MAPPING` HTTP header to the response. The expiration of the object is set based on the TTL contained with the mapping table. The body of the response is just the ESI augmented template. Once the cache has the template, it will go on and request the fragment as needed from the server. The cache can then build up the proper final document and send that off to the client. The servlet's support of the `If-Modified-Since` HTTP header when requesting the template is crucial to the efficiency of the mapping table. When the template expires on the cache (TTL equals or less than zero), the cache will request the template over again using the `If-Modified-Since` HTTP header. Since the template rarely changes, this will mean that the cache will usually get a '304 Not Modified' response. This response will contain the new mapping table, allowing the cache to update its mapping table without having to re-transfer the template. If a static document, or a static fragment is requested from the content delivery module, then the content delivery module will behave just like a regular Web server.

4.3 ESI Augmentation

As explained earlier, ESI provides and extensive range of existing technology to support a wide range of uses in structuring a template file and specifying things like TTL and so forth. In trying to remain as standard compliant as possible, the ESI format was picked to represent the structure of the template. Yet the ESI standard only supports document fragments identified by static URLs, which will not suffice in our case. Thus we needed to augment the ESI standard by adding the `esi:xconca-include` tag. This tag allows the specification of an ID number that can be looked up by the client in the mapping table and retrieve the object's static URL.

5 Performance Evaluation

The experimentation of our proposed method for object extraction and object delivery required simulating a fragment-enabled client-cache-server model. Using this model we can compare different types of performance for the different types of fragment-enabled dynamic content behaviors. The experiments we used to test the performance of the different caching systems targeted user-perceived latency and bandwidth usage specifically.

5.1 CONCA-Lite

The experiments in our client-cache-server model required a proxy that supported our proposed augmentation to the ESI and supported the dynamic assembly of the final content for the client. To achieve this, a simple cache proxy, called CONCA-Lite, was implemented using Python and its `asyncore` modules to create a simple extensible proxy. It was designed such that testing different caching methods would require little or no change on the proxy side. Thus, allowing us to make fair and accurate comparisons between the different caching methods, which are tested on the same caching framework. This CONCA-Lite proxy, which implemented a minimalistic version of the CONCA proposal [39], was then used to test the

different dynamic caching methods described later.

5.2 Evaluation Platform

We simulated the client-cache-server model using three machines, all connected on a 100Mbit/s LAN, at most, separated by a switch. The server, a 2.0 Hz Pentium 4 machine with 512 MB RAM with Linux, ran the Jigsaw Java server to host the DyCA servlets. The cache, a 2.4 Hz Pentium 4 machine with 1 GB RAM with Linux, ran the CONCA-Lite. The client, a 2.2 Hz machine with 512B of RAM with Linux, ran a Python-based simulation of a client accessing Web pages in a predefined order.

We modeled 4 different caching behaviors using our experimentation. To test each approach, the client was set up to request the Web page of the server from the cache at request intervals of 10 seconds, for a total of 10 minutes. The cache would check to see if it has the needed document, request the document from the server if it needs to, and return the document to the client. When testing a caching behavior that has a fragment enabled template, it would request all of the objects in the template, it would construct the final document, and return it to the client. Thus the client does not need to implement anything beyond the standardized HTTP protocol. The first method, using no fragment caching, was implemented by disabling caching in the proxy, and having the server send the original Web page. This is consistent with the behavior of real dynamic content using static pages in today's Internet, due to cookies, and other such information, that render a page uncacheable. In the second method the template of the document remains static, while the fragments of the objects are updated for content change. The template was cacheable for the whole testing session, while the objects were cacheable for as long as their TTL was valid. In the third caching behavior, the template is updated when a fragment moves between locations in the document, and the objects change due to data changes only, and not spatial changes. The last model represents our proposed mapping table approach. When the template is returned a mapping table is returned with it in the HTTP header, the proxy can then cache the mapping table, and update the mapping table when a spatial change happens. The template remained static for the testing session, while the objects only changed for data changes.

5.3 Experimentation

Three Web sites, New York Times [32], India Times [22], and Slashdot.com [40], were used for testing each approach. Two types of measurements were taken during the testing to evaluate the performance, the total amount of data transfered between the client and the server, and the user perceived latency per user request. The first type of measurement is important to show what method performs best as a cache, with the least data transfer between the cache and the server. There is no need to check for the data transfer between the client and the cache, since in all four models it should remain roughly the same. The generation of the final non-fragmented Web page by the cache that gets sent to the client, and the method it is updated, is what changes between each method. The second measurement type is important to show how an improved Web caching architecture will benefit the client as well, and not only the server.

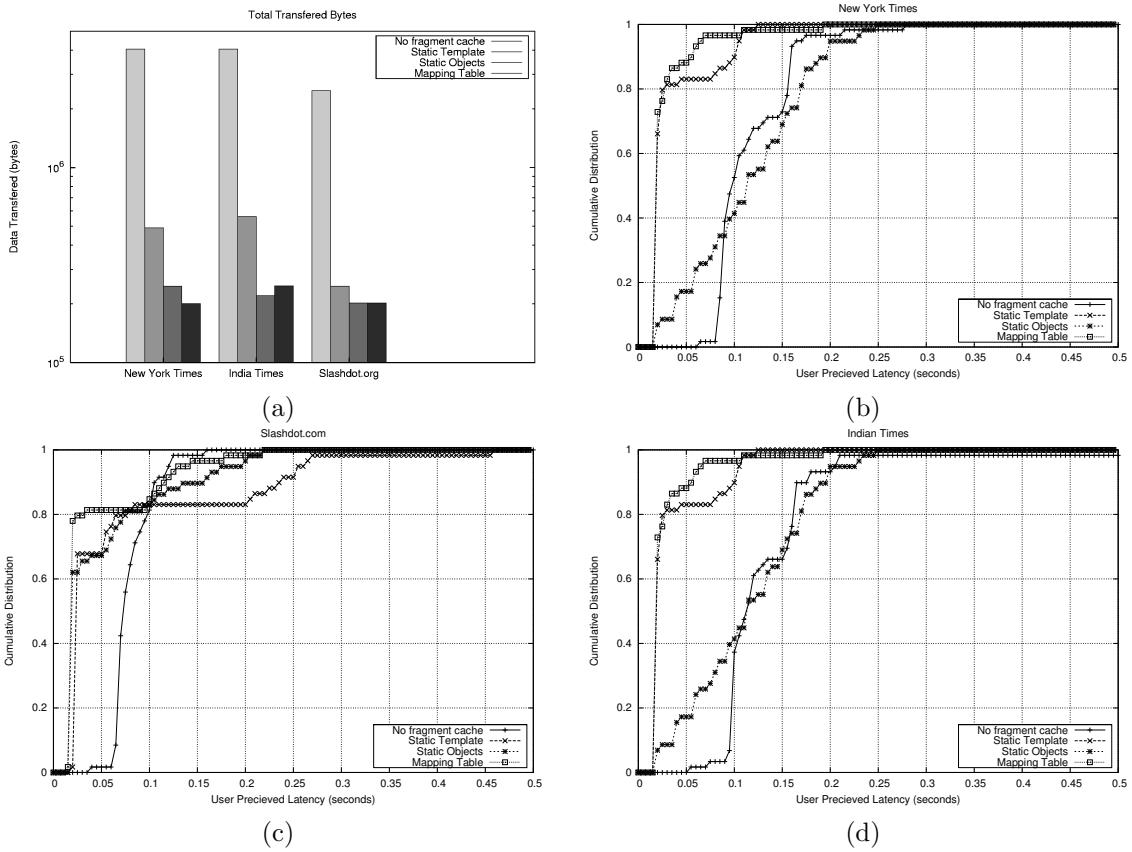

Fig. 7. Evaluation results: (a) total data transfer between server and cache, and user perceived latency for: (b) New York Times, (c) Slashdot, and (d) India Times.

5.4 Results and Analysis

Figure 7(a) demonstrates the total bytes used in transferring data in all of the methods mentioned. As can be seen from the graphs, using a static template and updating the objects to support object movements requires considerable more data transfer between the server and the proxy. This is the method that is most commonly used today in dynamic Web sites. Using static objects and a dynamic template to achieve a dynamic Web page might seem efficient enough when looking at the bytes transfered, but, as we will see later, this efficiency is lost when looking at the user perceived latency. There is no comparison, though, between any of the dynamic methods and using traditional static objects. The amount of data transfer when not using a fragment based architecture can be more then 10 times the amount of data transfer when using a good fragment based caching. When using a mapping table to transfer the data, the amount of bytes transfered is considerably smaller. In fact, the total bytes transfered with a mapping table is little over the total size of the initial site and the size of the changes, meaning a minimal amount of wasted data is being transfered. This should considerably reduce the server load when using such an architecture. Figures 7 (b)(c)(d), show the user perceived latency in seconds for New York Times, Slashdot, and India Times respectively. This figure shows that the user has to wait the least amount of time for the Web site when the mapping table architecture is used. With regard to user perceived latency, the static object method's performance is almost as bad as the performance of using regular static Web pages. The only method that comes close to the method of using a mapping table is the static template method, which as we saw before performed badly when looking at the amount of bytes transferred. This type of optimization is very important for the client so it may receive its data in a timely manner, especially clients that use slower connections such as dialup. Otherwise, from the users prospective, the actual retrieval of the page is slower. From these figures we can conclude that the mapping table method has performed better then all of the other proposed methods in all of our tested fields.

We can see some unaccounted behavior in how the 2nd and 3rd method flip in their efficiency between the results of the amount of bandwidth used, and the user latency done. When the template is static and there is no mapping table, the objects get transfered at a higher request rate since the proxy can't cache them due to object movement. This extra data transfer has little effect on the user perceived latency in our testing environment due to it being a high speed network. Yet this extra data transfer is significant in terms of amount of data transfered, as can be seen in the earlier figure. Since the only thing that needs to get updated every once in a while is the transferring of the template, in terms of latency, this is very close to transferring a mapping table at about the same interval. Yet by looking at Figure 8 you can see that the template is considerably larger then the mapping table in most cases. This is what causes the large amount of bytes to be transfered. Had we artificially slowed down the network, the user perceived latency for static objects would have been much greater. In the case of static objects, the templates is considered dynamic, and gets updated every time there an object moves. Every such template fetch requires the cache to re-parse the template, which can be very large as seen in Figure 8, and recheck it's cache for every single object, in some cases this requires the cache to send HTTP request to see if the data was modified. This type of overhead causes the extra latency seen in the graph.

	New York Times	India Times	Slashdot
Template Size	17 KB	15 KB	1.7 KB
Avg. Object Size	3.6 KB	4.8 KB	0.6 KB
Mapping Table Size	1.0 KB	0.8 KB	2.2 KB

Fig. 8. The comparison of template and object sizes for the different Web sites.

6 Related Work

Dynamic Web content delivery have been increasingly becoming an important Web engineering issue as more and more Web content are generated in a dynamic and personalized way. Fragment-based techniques have received considerable attention from the research community in recent years [12, 13, 16, 34, 44]. Most of these approaches either assume the fragment-based content is served by Web server automatically, or look at server-side caching only.

The notion of splitting up a dynamically generated personalized document into sharable and personalized components to improve cache effectiveness builds upon previous work done both at the server-side [12, 13, 14, 29, 49] and at the cache [10, 16, 28, 30, 44] to enable efficient generation and delivery of dynamic content. The common thread of these server-side mechanisms is that they all maintain dependence information, i.e., metadata about position and TTL of each fragment, between dynamically generated pages and the underlying data, and use this explicit dependence information to cache or incrementally update previously generated dynamic pages, reducing the cost of generating dynamic content. However, none of the existing cache architectures takes advantage of this dependence information to accelerate the delivery of dynamic content. Researchers have also suggested extending cache-side functionality to better handle dynamic content generation and caching, including *migrating application logic to caches* [10, 19, 30, 33, 36, 47], and *assembling dynamic content at caches*, such as HPP [16], content assembly [28], and client-side include (CSI) [34]. In both cases, only server-side information is taken into account, and client-side information is not used. A comparison between ESI and delta-encoding is provided in [31].

To our knowledge, few of existing work discuss the manner of how to generate fragment from existing legacy Web servers without server-side information. One of the first effort is DYCE [37], which is model-based dynamic Web content emulator. Recently, Ramaswamy *et al.* proposed a novel scheme to automatically detect and flag fragments [35], which shares the similar goal of this paper. However, there are three differences between us: First, although both of our work intends to automatic detection of fragments, our keyword-based is simple and easy to implement, while their approach is complex and has theoretical analysis. However, which one is better is still not clear. Second, in our work we focus on engineering implementation of DYCA, while their work focuses on automatic detection. In this sense, their work is a good complement to DYCA. Third, the mapping table based fragment delivery proposed in this paper is novel.

Our current research differentiates from earlier work done on DYCE [37], the Dynamic Web Content Emulator. With DYCE, we were attempting to build up general models for usage to describe the behavior of current fragment-based caching. Although it looked promising at the time, it generated too many objects and didn't match up to actual real world designs. The theoretical results in DYCE give out an upper bound for fragment based caching, while in this work, the real engineering and implementation issues and detailed performance evaluation

are the primary of concern. Moreover, the fragment extraction techniques used in these two approaches are different. In DyCA, a keyword-based heuristic approach is proposed, while size-based and tree depth based heuristic schemes were used in DYCE. We believe that the keyword-based approach is more realistic. As an nature extension of DYCE, our current research is an attempt to try and continue that same research using real world Web sites so as to get more correct and realistic results.

Edge Side Includes [42] is becoming one of the foundation blocks in specifying a common format and method for fragments and templates in this field. It is popular among many different existing methods. Naaman *et al.* [31] have done studies comparing ESI to delta encoding, finding ESI to have possible performance advantages.

Automatic detection of templates from Web pages has been studied from data mining field as well [3, 5]. They discuss the problem of template detection through discovery of pagelets in the Web pages. However, our work differs from the work on template detection both in context and content. First, the work on template detection is aimed towards improving the precision of search algorithms. While our aim is improving dynamic content delivery. Second, only template is interested in their work, while we care both template and fragments. Therefore, the method used in these two approaches are different too. The method presented there to finding fragments is done based on amount of hyperlinks present in certain parts of the document. They do not build up an XML tree, nor treat anything more then hyperlinks, unlike we have done in our approach. This method applies better to search algorithms rather then to dynamic fragment extraction. Semantic Web information annotation and extraction is also a hot research topic in Semantic Web community [24]. The work in this community shares the similar goal of our work to automatically extract metadata of Web content; however, the difference is in the granularity of information of interest. In semantic information extraction, the granularity of interest is relatively small, such as words, phrases, etc., while in this paper we care more about coarse grain fragments because fine grain fragments will not bring any benefit for caching purpose. However, we believe the work in sematic web community compliments to our work, and we plan to leverage their results to extracting fragments in the future.

Other research groups [8, 20] have also defined other criteria for finding objects in documents. While they have focused on content of the fragments and of the Web pages themselves, we have focused on their existence on a spatial, and location axis in the document.

Commercial caching and edge server products, most notably IBM's WebSphere [21] and Akamai's EdgeSuite [15] are beginning to support dynamic Web content caching within their products by using ESI [42]. However, their solutions are server-oriented and not focused on intermediary cache-level support, which is worthwhile pursuing because it provides the advantage that cache policies can be better tailored to the supported user base. Also, due to their proprietary settings, the performance of their approach is unavailable.

7 Conclusions and Future Work

We are witnessing the fast growth of dynamic Web services resulted from the advance of Web engineering. In this paper, we argue that finding an effective way to deliver these dynamic Web services is a very important Web engineering issue. In the context of Web caching, we have shown our proposed solution for keyword based object extraction, and object delivery. We

have also explained our proposal of augmenting the ESI to include support for a mapping table. We have implemented these proposals into DyCA and then by taking actual Web pages and running through DyCA's keyword-based extraction to transform them into fragment-enabled content we have been able to run simulations between our sample proxy and the DyCA adapter. These simulations allowed us to compares our proposals to the current available methods of serving dynamic content on the Web. Our keyword-based approach allows for creation of dynamic content in such a way as to maximize the cache-ability of the content in a fragment-enabled caching system. Using the mapping table approach in the cache proxy which, according to our results, will give the best performance for both the server and the client, together with our DyCA adapter we can effectively cache in an efficient way Web sites that currently use non-fragmented content.

Currently our DyCA and CONCA implementations are at the prototype stage, and could still be further optimized. Our future work consists of continuing testing of these implementations to further refine the design of our CONCA prototype [39], and make the code available in open source. Another direction to extend our work will be examining the function caching, i.e., moving the functionality from server side to the network edge, and the relationship between function caching and Web application design, such as component-based Web engineering [18].

Acknowledgements

We are grateful to our guardian Bebo White for his constructive comments. We also thanks the anonymous ICWE'04 reviewers for helping us improve the presentation of this paper.

References

1. Akamai Technologies Inc., `http://www.akamai.com/`.
2. Apache HTTP Server Project, `http://httpd.apache.org`.
3. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. *Proc. of ACM SIGMOD'03*, pp. 337-348, June 2003.
4. A. Awadallah and M. Rosenblum. The vMatrix: A network of virtual machine monitors for dynamic content distribution. *Proc. of the 7th International Workshop on Web Caching and Content Distribution (WCW'02)*, Aug. 2002.
5. Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. *Proc. of the 11th International World Wide Web Conference (2002)*, pp. 580-591, May 2002.
6. R. Belotti et al. Interplay of content and context. *Proceedings of the 4th International Conference on Web Engineering (ICWE'04)*, pp. 187-200, July 2004.
7. L. Bent, M. Rabinovich, G. Voelker, and Z. Xiao. Characterization of a large web site population with implications for content delivery. *Proc. of the 13th International World Wide Web Conference (2004)*, pp. 522-533, May 2004.
8. D. Butler and L. Liu. A Fully Automated Object Extraction System for the World Wide Web. *Proceedings of ICDCS-2001*, 2001.
9. R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web proxy caching: The devil is in the details. *Proceedings of ACM SIGMETRICS Internet Server Performance Workshop*, June 1998, `http://www.douglis.org/fred/work/papers/wisp98.ps`.
10. P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents on the web. *Proc. of IFIP Int'l Conf. Dist. Sys. Platforms and Open Dist. Processing*, pp. 373-388, 1998, `http://www.cs.wisc.edu/~cao/papers/active-cache.ps`.

11. S. Ceri, P. Dolog, M. Matera, and W. Nejdl. Model-driven design of web applications. *Proceedings of the 4th International Conference on Web Engineering (ICWE'04)*, pp. 201-214, July 2004.
12. J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic web data. *Proc. of IEEE Conference on Computer Communications (INFOCOM'99)*, Mar. 1999.
13. J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. A publishing system for efficiently creating dynamic web content. *Proc. of IEEE Conference on Computer Communications (INFOCOM'00)*, Mar. 2000.
14. A. Datta, K. Dutta, H. Thomas, D. VanderMeer, and K. Ramamritham. Accelerating dynamic web content generation. *IEEE Internet Computing* 6(5):26–35, September/October 2002.
15. J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing Magazine* 6(5):50–58, September/October 2002.
16. F. Douglis, A. Haro, and M. Rabinovich. HPP:HTML macro-pre-processing to support dynamic document caching. *Proc. of the 1st USENIX Symposium on Internet Technologies and Systems (USITS'97)*, pp. 83-94, Dec. 1997, `http://www.douglis.org/fred/work/papers/hpp.pdf`.
17. A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. *Proc. of IEEE Conference on Computer Communications (INFOCOM'99)*, pp. 107-116, Mar. 1999, `http://www.douglis.org/fred/work/papers/hetproxcache.pdf`.
18. M. Gaedke and J. Rehse. Supporting compositional reuse in component-based web engineering. *Proceedings of the ACM Symposium on Applied Computing*, pp. 101-106, 2000.
19. L. Gao, M. Dahlin, A. Nayate, and A. Iyengar. Application specific data replication for edge services. *Proc. of the 12th International World Wide Web Conference (2003)*, May 2003.
20. X. Gu et al. Visual based content understanding towards web adaptation. *Proceedings of AH-2002*, 2002.
21. IBM Corp. Websphere platform, `http://www.ibm.com/websphere`.
22. Indiatimes web site, `http://www.indiatimes.com`.
23. Jigsaw Project, `http://www.w3.org/Jigsaw`.
24. A. Kiryakov et al. Semantic annotation, indexing, and retrieval. *Journal of Web Semantics* 2(1), 2005.
25. B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. *Proceedings of SIGCOMM IMW 2001*, pp. 169-182, Nov. 2001, `http://www.research.att.com/~bala/papers/imw01-abcd.pdf`.
26. U. Manber, A. Patel, and J. Robison. Experience with personalization on Yahoo! *Communications of ACM* 43(8):35–39, Aug. 2000.
27. V. Mastoli, V. Desai, and W. Shi. SEE: a service execution environment for edge services. *Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP'03)*, pp. 61-65, June 2003.
28. M. Mikhailov and C. E. Wills. Change and relationship-driven content caching, distribution and assembly. Tech. Rep. WPI-CS-TR-01-03, Computer Science Department, WPI, Mar. 2001, `http://www.cs.wpi.edu/~cew/papers/tr01-03.pdf`.
29. J. C. Mogul, F. Douglis, a. Feldmann, and B. Krishnamurthy. Potential Benefits of Delta-Encoding and Data Compression for HTTP. *Proc. of the 13th ACM SIGCOMM'97*, pp. 181-194, Sept. 1997, `http://www.douglis.org/fred/work/papers/sigcomm97.pdf`.
30. A. Myers, J. Chuang, U. Hengartner, Y. Xie, W. Zhang, and H. Zhang. A secure and publisher-centric web caching infrastructure. *Proc. of IEEE Conference on Computer Communications (INFOCOM'01)*, Apr. 2001.
31. M. Naaman, H. Garcia-Molina, and A. Paepcke. Evaluation of esi and class-based delta encoding. *Proc. of the 8th International Workshop on Web Caching and Content Distribution (WCW'03)*, Sept. 2003.
32. Nytimes web site, `http://www.nytimes.com`.
33. M. Rabinovich, Z. Xiao, and A. Aggarwal. Computing on the edge: A platform for replicating internet applications. *Proc. of the 8th International Workshop on Web Caching and Content Distribution (WCW'03)*, Sept. 2003.

34. M. Rabinovich, Z. Xiao, F. Douglis, and C. Kamanek. Moving edge side includes to the real edge – the clients. *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, Mar. 2003.
35. L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis. Automatic detection of fragments in dynamically generated web pages. *Proc. of the 13th International World Wide Web Conference (2004)*, pp. 443-454, May 2004.
36. J. Ravi, W. Shi, and C. Xu. Personalized email management at network edges. *IEEE Internet Computing* 9(2), March/April 2005.
37. W. Shi, E. Collins, and V. Karamcheti. DYCE: A synthetic dynamic web content emulator. *Poster Proc. of 11th International World Wide Web Conference*, May 2002, `http://www.cs.wayne.edu/~weisong/papers/dyce.pdf`.
38. W. Shi, E. Collins, and V. Karamcheti. Modeling object characteristics of dynamic web content. *Journal of Parallel and Distributed Computing* 63(10):963–980, Oct. 2003.
39. W. Shi and V. Karamcheti. CONCA: An architecture for consistent nomadic content access. *Workshop on Cache, Coherence, and Consistency(WC3'01)*, June 2001, `http://www.cs.wayne.edu/~weisong/papers/wc301.pdf`.
40. Slashdot web site, `http://www.slashdot.com`.
41. M. Taguchi et al. Comparison of two approaches for automatic construction of web applications: Annotation approach and diagram approach. *Proceedings of the 4th International Conference on Web Engineering (ICWE'04)*, pp. 230-243, July 2004.
42. M. Tsimelzon, B. Weihl, and L. Jacobs. ESI language specification 1.0, 2000, `http://www.esi.org`.
43. C. E. Wills and M. Mikhailov. Towards a better understanding of web resources and server responses for improved caching. *Proc. of the 8th International World Wide Web Conference (1999)*, May 1999.
44. C. E. Wills and M. Mikhailov. Studying the impact of more complete server information on web caching. *Proc. of the 5th International Workshop on Web Caching and Content Distribution (WCW'00)*, 2000.
45. A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. *Proc. of 17th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 16-31, Dec. 1999.
46. W3C XSL Working Group, `http://www.w3.org/Style/XSL/`.
47. C. Yuan, Y. Chen, and Z. Zhang. Evaluation of edge caching/offloading for dynamic content delivery. *Proc. of the 12th International World Wide Web Conference (2003)*, May 2003.
48. C. Yuan, Z. Hua, and Z. Zhang. Proxy+: Simple proxy augmentation for dynamic content processing. *Proc. of the 8th International Workshop on Web Caching and Content Distribution (WCW'03)*, Sept. 2003.
49. H. Zhu and T. Yang. Class-based cache management for dynamic web content. *Proc. of IEEE Conference on Computer Communications (INFOCOM'01)*, Apr. 2001, `http://www.cs.ucsb.edu/projects/swala/cache2001.ps`.
50. Z. Zhu, Y. Mao, and W. Shi. Workload characterization of uncacheable web content. *Proceedings of the 4th International Conference on Web Engineering (ICWE'04)*, pp. 391-395, July 2004.