Proxy+: Simple Proxy Augmentation for Dynamic Content Processing

Chun Yuan[1] t-cyuan@microsoft.com
Zhigang Hua[2] zghua@mails.gscas.ac.cn
Zheng Zhang[1] zzhang@microsoft.com

[1]Microsoft Research Asia
[2]Institute of Automation, Chinese Academy of Sciences

Abstract

Caching dynamic content can bring many benefits to the performance and scalability of Web application servers. However, such mechanisms are usually tightly coupled to individual application servers (or even applications) that prevent caching at more advantageous points. In this paper we propose an approach to enable dynamic content caching at enhanced Web proxies which requires only simple modifications to existing applications.

1 Introduction

Dynamic content will dominate the Web in the future. This necessitates architectural change in tandem. In particular, resources that are already deployed near the client such as the proxies that are otherwise underutilized for such content should be employed.

Legitimate strategies include offloading some of the processing to the proxy, or simply enhancing its cache abilities to cache fragments of the dynamic pages and perform page composition. While performance benefits including latency and server load reduction are important factors to consider, issues such as engineering complexity as well as security implication are of even higher priority. Our previous work [20] investigates what will be the best offloading and caching strategies and their design/deployment tradeoffs given the proxy resources at the edge of the network. We have shown that simply caching dynamic page fragments and composing the page at the proxy achieve close performance to other strategies. In fact, advanced offloading strategies can be overly complex and even counter-productive performance-wise if not done carefully.

This work continues the previous work to demonstrate the advantage and feasibility of proxy caching for dynamic content. We propose Proxy+ by adding a caching filter at Microsoft ISA proxy server [14] and offers fragment caching and page composition ability. Our core idea is to simply replicate the server-side caching functionality to the proxies, while carefully engineering the protocols so that consistency enforcement takes a free-ride from what the programmers have already expressed when enabling server-side dynamic content caching. Our architecture requires only minor modifications to existing applications and is incrementally deployable. Finally, although our prototype is implemented under Microsoft ASP.NET [13] and ISA Server, we believe the method is equally applicable to other platforms.

The rest of the paper is organized as follows. Section 2 covers related work. Section 3 summarize our previous results which motivates this work. Section 4 describes the architecture, components and protocol of Proxy+. Section 5 illustrates how existing ASP.NET applications can be modified to work with Proxy+. Section 6 reports our experimental results. Section 7 discusses some security issues. Finally we conclude in section 8.

2 Related Work

Optimizing dynamic content processing has been widely studied. Most work focused on supporting dynamic content caching on server side [9][10][19]. Currently there are already mature application server products offering this feature, e.g. Microsoft ASP.NET, BEA WebLogic [1], IBM WebSphere [7], Oracle9iAS [16]. Server-side caching can reduce server load and therefore improve response time when client stress is high. Moving the caching tier to proxies at the network edge that is closer to clients would bring more benefits as reported in [11][20].

Fragments caching is key to dynamic content caching, since it can improves page cacheability and cache efficiency. Many Java application servers allow programmers to mark a part of a page as cacheable using JSP tags. In ASP.NET such fragment can be explicitly put into a user control which has its own cache parameters and can be included by pages or other user controls. The cache

is usually associated with the application server, thus preventing caching at more advantageous points. [5] also uses tags to support fragment caching on a reverse proxy which is assumed to be near the server, but it does not consider caching on remote proxies. Active Cache [3] is a rather general scheme to push server-side processing to proxies but it does not address specific issues with fragment caching. IBM WebSphere's Trigger Monitor [8] also support fragment caching, but on predetermined edge servers only.

ESI [4] proposes to cache fragments at the CDN stations to reduce network traffic and response time. ESI introduces some directives for programmers to author cacheable Web pages which will be interpreted by ESI engine on edge servers. A page may include fragments which will be retrieved separately from the server (when cache miss). Therefore for existing applications to employ ESI, original single output of a page has to be divided into parts which can be separately generated and retrieved. The top-level page output also need to contain ESI include directive for edge servers to get inner fragments. However this could violate original request processing workflow and semantics, because during original page generation the top-level page and the fragments are in the same request context while after factoring they are independently requested. Hence turning a fragment into a page would require understanding of the original page semantics and may require considerable reengineering effort.

Proxy+ relies on output tagging to distinguish fragments so that they can be independently cached. It requires only trivial extension to existing applications, provided that they have already made use of fragment caching on server side. Furthermore it does not interfere with the application's original workflow and need not to understand the details of the application. We let proxies notify a list of cache keys that are deemed relevant to the request so that network traffic and server load can be saved as much as possible.

3 Summary of Previous Result

In our previous work [20] we studied the performance of four Web application configurations, with three of them taking advantage of edge proxy to offload server processing.

- F_0: the default setting, with all processing happening on the server
- $F_{remoting}$: push part of application logic (e.g. presentation tier) to the proxy
- F_{db}: push everything except database to the proxy
- F_{proxy}: push fragment caching and page composition functionality to the proxy

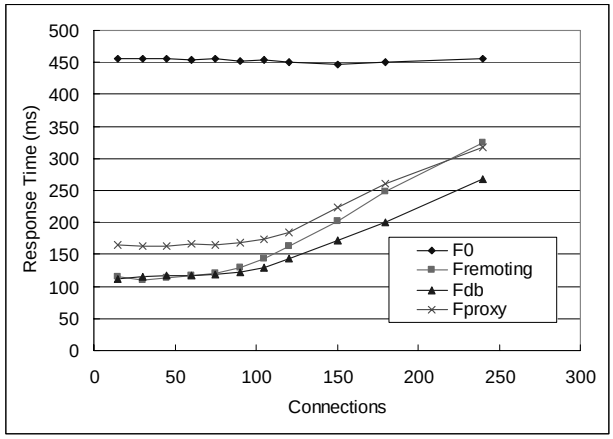


Figure 1. Response time versus number of connections. The roundtrip time is 400ms.

Our results (Figure 1) show that under typical user browsing patterns and network conditions, 2~3 folds of latency reduction can be achieved. Furthermore, over 70% server requests are filtered at the proxies, resulting significant server load reduction. Interestingly, this benefit can be achieved largely by F_{proxy} – simply caching dynamic page fragments and composing the page at the proxy. Taking implementation cost and security into account, our result can be summarized in Table 1.

In other words, augmenting today's proxy caching capability has the best tradeoff. The F_{proxy} implementation in that paper took an ESI-like approach and was application-dependent, which

	Performance	Security req.	Complexity
F_{db}	Best	High	Low
$F_{remoting}$	Second	Unsure → High	High
F_{proxy}	A close 3rd	Low	Lowest

Table 1. Comparison of different configurations

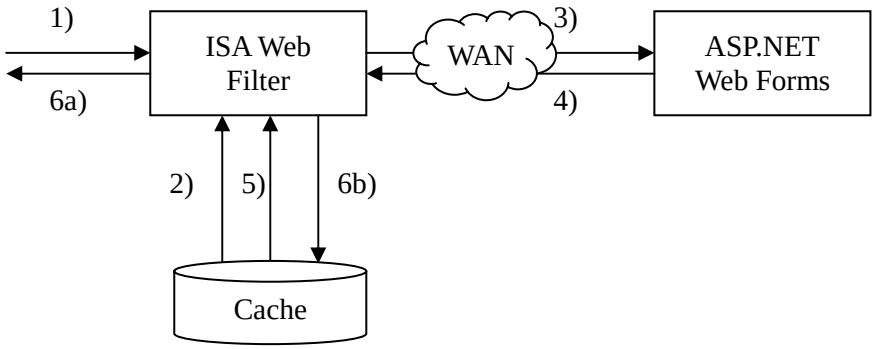


Figure 2. Architecture and workflow of Proxy+

motivates the work of this paper.

4 Proxy+ Architecture

Given that proxy augmentation offers the best tradeoff, we developed the prototype of Proxy+ on top of Microsoft ISA Server. The augmented proxy is able to do caching for ASP.NET applications which have made use of ASP.NET built-in output caching facility after minor modifications. In fact, these modifications will be trivial if necessary supports are absorbed by ASP.NET.

The core idea is straightforward: replicate the output cache functionality at the proxy with an ISA Web filter. Therefore, the engineering focus is on how to get a "free ride" by mirroring caching semantics and its enforcement from server side's output cache. Figure 2 shows the architecture and workflow of Proxy+.

The ISA Web Filter is responsible for directing the caching of multiple versions of pages and fragments as well as composing pages. Cache is the storage of previously requested pages and fragments. Both of them reside on the ISA Server. With ASP.NET, Web Forms constitute the presentation tier of the Web application running on the server side. In our framework, applications can cooperate with the ISA Server to take advantage of its enhanced caching ability after minor extensions to the classes that the Web Forms inherit from.

The system workflow is described as follows.

1) An HTTP request arrives at the proxy.
2) The filter computes the cache keys of the page and its fragments.
3) If all necessary items are valid in the cache, go to 5) and the result will be returned immediately. Otherwise, it attaches a list of keys identifying cached versions deemed relevant to the HTTP request header and forwards it to the server.
4) The application generates a (partial) response containing additional tags for delimiting cacheable fragments or behaving as placeholders to be substituted with cached content. In addition, necessary information, in the form of *Cache Variation Logic* (CVL) tags, which allows the proxy to compute the cache keys are sent over.
5) The filter parses the content (from the response or the cache) and fills the placeholder tags in the text with corresponding cached content, and installs any CVL tags. This way, CVLs are incrementally pulled over on-demand.

6a) A complete response is sent back.
6b) The fragments marked for caching are saved to the cache.

In the following text, we are going to illustrate the essential components in turn:

- We will start by reviewing ASP.NET's output cache and in particular its semantics of caching multiple versions.
- We will then present how cache keys are computed with the help from the application side. This entails two steps: modification at application to generate CVL tags, and proxy-side's use of such tags to produce cache keys.
- Next, we describe how the proxy assembles a page out of content in its cache and, if necessary, notify the server/application what are already available in its cache to avoid redundant transfer.

4.1 ASP.NET output caching

The presentation tier of an ASP.NET Web site consists of a set of Web Form pages (with .aspx extension) and user controls (with .ascx extension). They are responsible for generating Web pages (usually in HTML) to satisfy client requests. A user

control represents a fragment of a page and can be included by many pages or other user controls.

ASP.NET allows many versions outputted by a page to be cached (at the server side). Programmers can use a high-level, declarative API or a low-level, programmatic API when manipulating the output cache. For example, the following @OutputCache directive (included in an .aspx or .ascx file) sets an expiration of 60 seconds for the cached output of the page or user control.

```
<%@ OutputCache Duration=”60” VaryByParam=”none” %>
```

Upon arrival of subsequent requests, the output cache will send the proper cached version as response directly. While this is the basic version, advanced features require programmers to specify how the output cache is varied by. @OutpputCache directive includes the following attributes referred to as Cache Variation Logic (CVL) that can be used to cache multiple versions of page output.

- **VaryByParam**: vary the cached output depending on GET query string or form POST parameters. Including the following example at the top of an .aspx file will cause different versions of output to be cached for each request that arrives with a different “city” parameter value.

  ```
  <%@ OutputCache Duration="60" VaryByParam="city" %>
  ```

- **VaryByHeader**: vary the cached output depending on the HTTP header associated with the request. The following example sets versions of a page to be cached, based on the value passed with the “Accept-Language” HTTP header.

  ```
  <%@ OutputCache Duration="60" VaryByParam="none" VaryByHeader="Accept-Language" %>
  ```

- **VaryByCustom**: vary the cached output by a custom string. This is the most advanced and powerful option. A special method called HttpApplication.GetVaryByCustomString() must be overridden which is used to map a string name to a value under some context. The following directive will cause the output to be cached for each request with a different value corresponding to “userstatus”, which may be, for example, “login” or “logout”.

  ```
  <%@ OutputCache Duration="60" VaryByParam="None" VaryByCustom="userstatus" %>
  ```

ASP.NET output cache computes a key based on CVL, and searches for the associated content in the cache. The key calculation is *internal* to the output cache. Consequently, proxy-side cache must be keyed with its own algorithm.

4.2 Cache key generation

In order to enable proxy-side output caching, the proxy must uses its own key generation algorithm. Our current implementation simply concatenates with semicolons the path name of the page (or user control) and the values that the output depends on in order to produce the cache key. For example suppose the page http://www.petshop.net/Category.aspx has the following CVL.

```
<%@ OutputCache Duration="60" VaryByParam="category_id" VaryByHeader=”Accept-Language” %>
```

If it receives a request http://www.petshop.net/Category.aspx?category_id=cats, and the header field “Accept-Language” is “zh-cn”, then the cache key is “/Category.aspx;cats;zh-cn”.

And suppose the CVL of the user control named “header” at http://www.petshop.net is as follows.

```
<%@ OutputCache Duration="60" VaryByCustom=”userstatus” %>
```

It is included by “/Category.aspx” and is intended to show different interface for anonymous users and authenticated users. When requested, the programmer-defined method GetVaryByCustomString() will analyze the cookie and decide the user status. If the user has signed in, it will map “userstatus” to, for example, “login”. Therefore, the final cache key of header’s output would be “header;login”. The way for the proxy to get the method is described in the next section.

In Proxy+, to avoid redundant computation and transfer, whatever contents available at proxy will be notified to the server, using the cache keys. Server runs the same key generation algorithm to generate the cache keys again so as to skip the redundant content generation. This will be described in more detail in section 5.

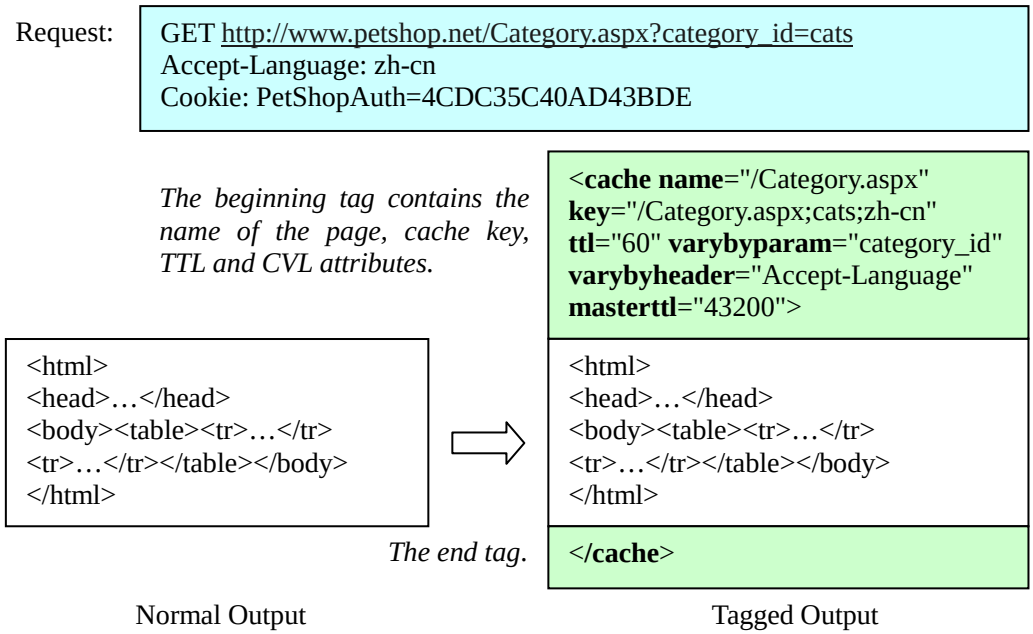


Figure 3. Tagging the output of a page

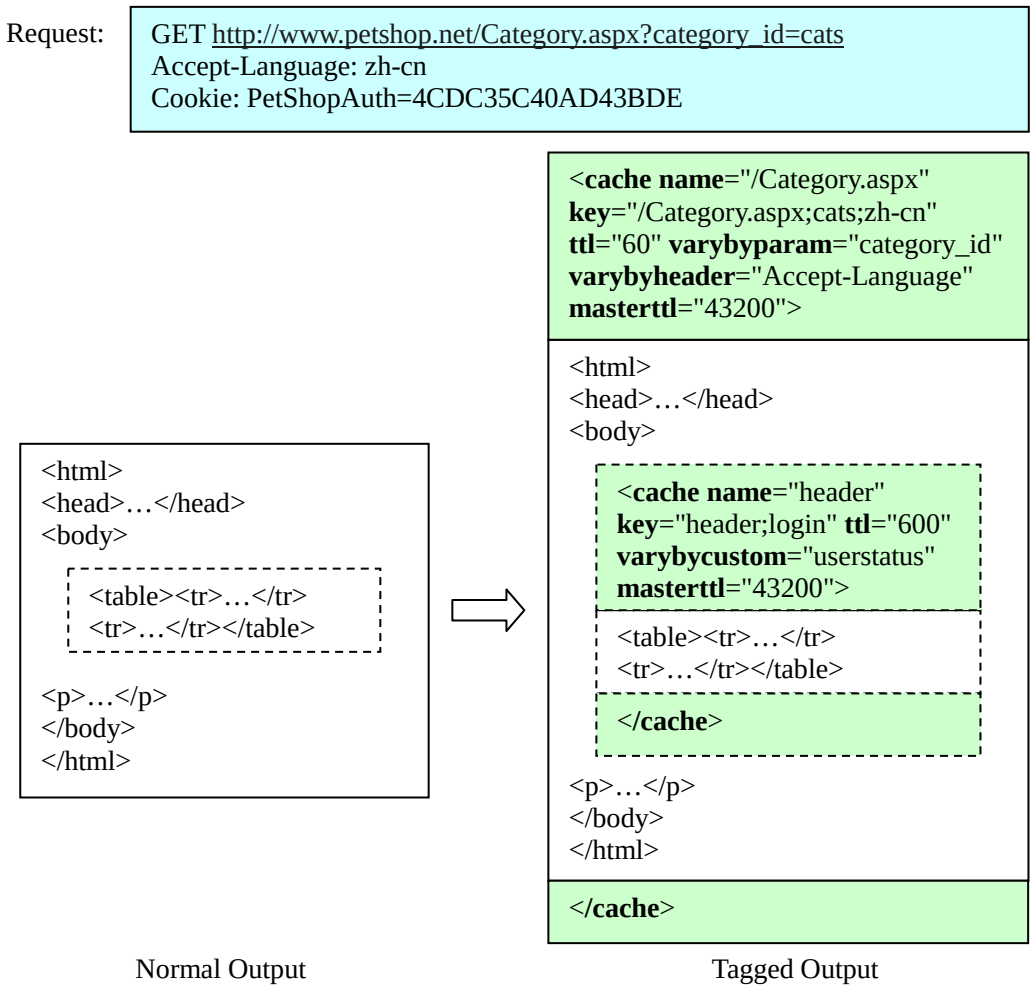


Figure 4. Tagging the output of a page as well as its inner fragment

Of course, in practice a proxy may interact with many servers, so the scope of a cache key is limited to the server which defines the CVL. In the above case, the cache key "/Category.aspx;cats;zh-cn" and "header;login" are only applicable to requests sent to the server www.petshop.net.

4.3 Tag generation and fragment caching

The CVL and TTL (time to live, i.e., Duration attribute in @OutputCache directives) of a page is communicated to the proxy through additional `<cache>` tags in the output. Figure 3 illustrates how such tags are added to the normal output.

When receiving a tagged response, the proxy will cache the actual content with the key. It also recognizes the CVL attributes according to which cache keys for subsequent requests will be calculated. The CVL is installed if it sees it the first time, or updated if necessary. This way, CVL tags are pushed to proxy incrementally in an on-demand fashion. The "**masterttl**" attribute declares the lifetime of the master program that produces the output, while "**ttl**" defines the lifetime of the specific version of output only. Therefore in the above example, the CVL of the page named "/Category.aspx" is valid for 12 hours.

Fragments are treated in the same way, though a tagged fragment might be contained in another tagged page or fragment. In this case, the outer page needs to cache the content at its level and the positions and names of the inner fragments. That is, the cached page output doesn't include the content of the fragments inside but their places and names instead, which are used to insert new versions of the fragments. Figure 4 shows how a page output containing a fragment is tagged (assume now "/Category.aspx" contains a fragment called "header" inside). Note that the inclusion relationship is also valid for 12 hours as limited by the "**masterttl**" attribute.

The page and the fragment will be cached as shown in Figure 5. The `<include>` tag represents a placeholder that is to be replaced with the content of a fragment with the specified name. All `<cache>` tags will be removed from the output and the client will receive a response just like the normal output.

As for the fragment "header" whose CVL containing **VaryByCustom**, it is not enough to notify the attribute values to the proxy because it depends on the function GetVaryByCustomString() to generate their cache keys. In Proxy+, the Web application specifies the location of the dynamic linking library by sending a new HTTP header, "X-GetVaryByCustom", with the response:

```
X-GetVaryByCustom: library-url
```

which must export the function:

```
string
GetVaryByCustomString(HttpRequest
req, string varyby);
```

Proxy is responsible for passing the complete request (the argument "req", including various header fields, cookies and body) and the value of **VaryByCustom** attribute associated with a fragment (the argument "varyby") to the function, which will return a unique string for identifying the version of the fragment. For example, besides generating the tagged output, the application would also add the following header to the response:

```
X-GetVaryByCustom:
http://www.petshop.net/varybycustom
```

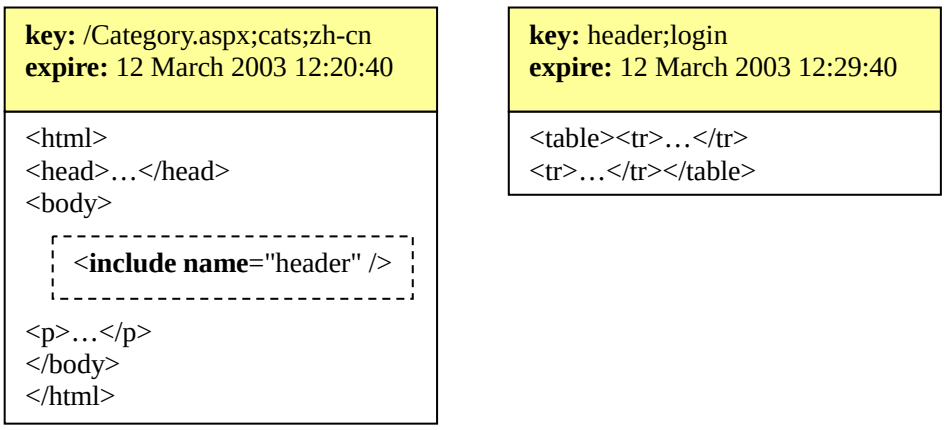


Figure 5. Cached output

`.dll`

The proxy can download the DLL and import the function. Requesting pages/fragments with **VaryByCustom** attributes will cause cache miss when the DLL is not available (not downloaded or become obsolete).

When receiving a subsequent request, if any of the necessary keys is not found in the cache (i.e., cache miss), the proxy will forward the request to the server. Otherwise it will compose the items corresponding to the keys together and return a complete response. Continuing the example, when receiving the request http://www.petshop.net/Category.aspx?category_id=cats again (with "Accept-Language" header being "zh-cn") the proxy computes a key "/Category.aspx;cats;zh-cn" that would be found in the cache. Then it computes another key for the fragment the page needs to include. If the user hasn't signed out, GetVaryByCustomString(Request, "userstatus") will return a string "login" according to the authentication cookie in the request and thus the key would be "header;login", which means both items hit the cache. The proxy will insert the content of "header" into that of "/Category.aspx" and the full output is returned. If the user has signed out and the cache doesn't contain the key "header;logout", the request will be forwarded to the server.

GetVaryByCustomString() may not be able to accomplish the same function on the proxy as its counterpart on the server when it requires server-specific resources like database. For example on a personalized portal site, the method may read user settings and produce a news type id (such as "science") that the user is interested in for a news fragment, while user settings are not accessible from proxies. Therefore not all kinds of pages and user controls with **VaryByCustom** attributes on the server are equally cacheable on the proxy.

4.4 Cache keys notification and page composition

Unlike traditional HTTP caching proxies, proxy+ can cache some parts of the response even when the others miss. If the application could know what parts have been cached on the proxy and output only those that are missing, server resources would be saved. Therefore, in our architecture a proxy is allowed to notify the server a list of keys along with the request to indicate that the page/fragments with these keys have been cached so that the server doesn't need to generate the content again. The notification is done by appending a new HTTP header field, "X-CachedKeys", to the incoming request:

```
X-CachedKeys: cache-key1,
cache-key2, ...
```

If the server-side application finds that the cache

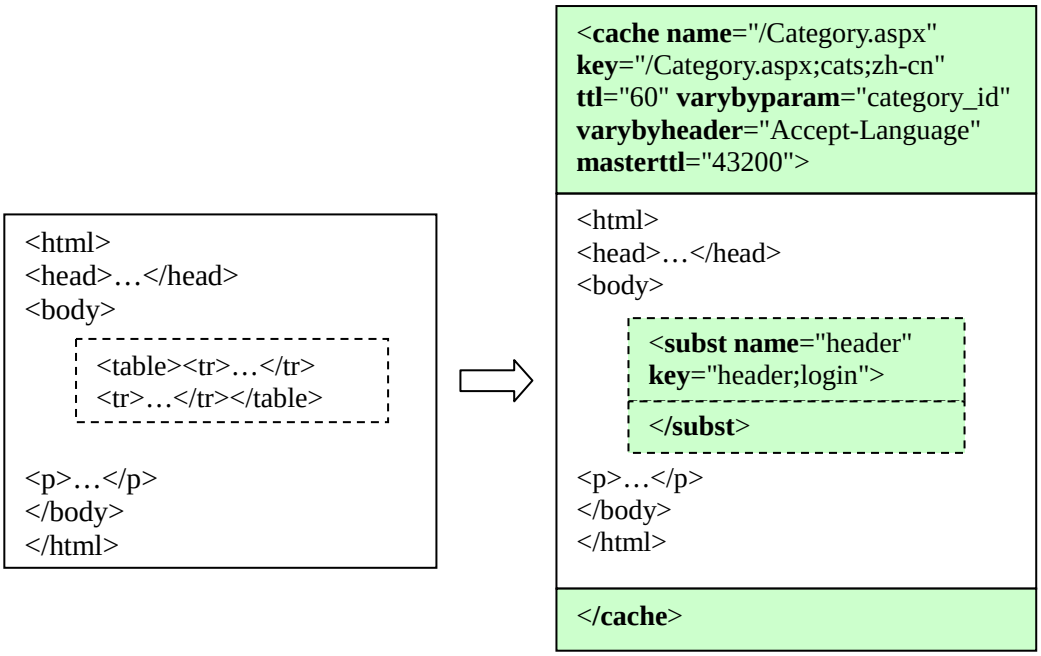


Figure 6. Using `<subst>` tag in place of cached inner fragment

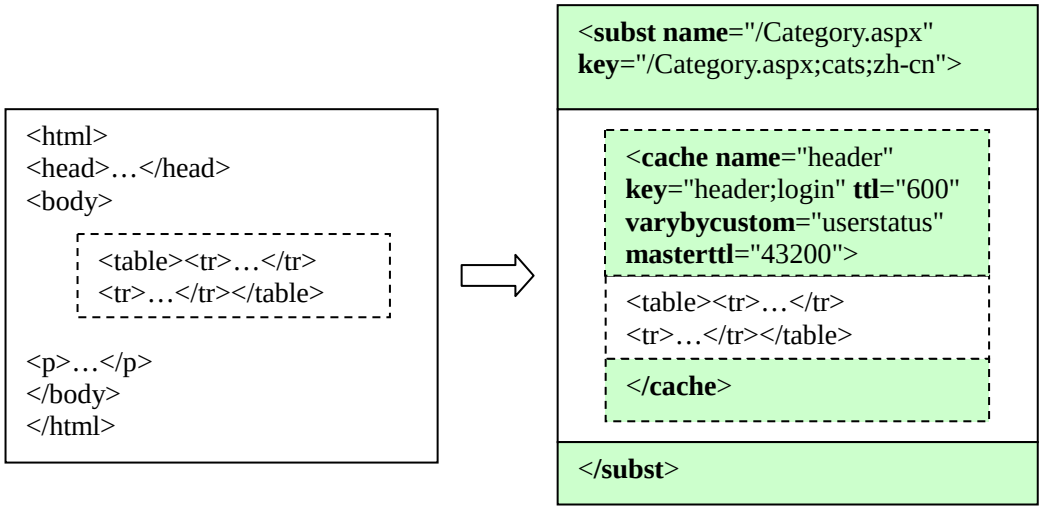


Figure 7. Using `<subst>` tag in place of cached outer page (fragment)

key of the page or fragment is listed in the header, it can skip the content generation process and instead put a placeholder tag (`<subst>`) along with the name and the cache key. The verification is processed by running exactly the same key generation algorithm as in proxy. Placeholder tags are intended to be substituted with the corresponding content from the proxy cache. For example, assume the proxy forwards the request http://www.petshop.net/Category.aspx?category_id=cats (with "Accept-Language" header equaling "zh-cn") with such a header added,

```
X-CachedKeys: header;login
```

If the user has signed in, the application will verify that the cache key of the inner fragment "header" is in the request header and can mark the output as in Figure 6. The pair of `<subst>` tag is to be replaced with the cached fragment having the key "header;login" on the proxy.

On the other hand, if the request has this header,

```
X-CachedKeys:
/Category.aspx;cats;zh-cn
```

the output can be like Figure 7.

On the proxy the fragment "header" will be inserted into the cached output with key "/Category.aspx;cats;zh-cn" in the position of the placeholder and a complete page returned.

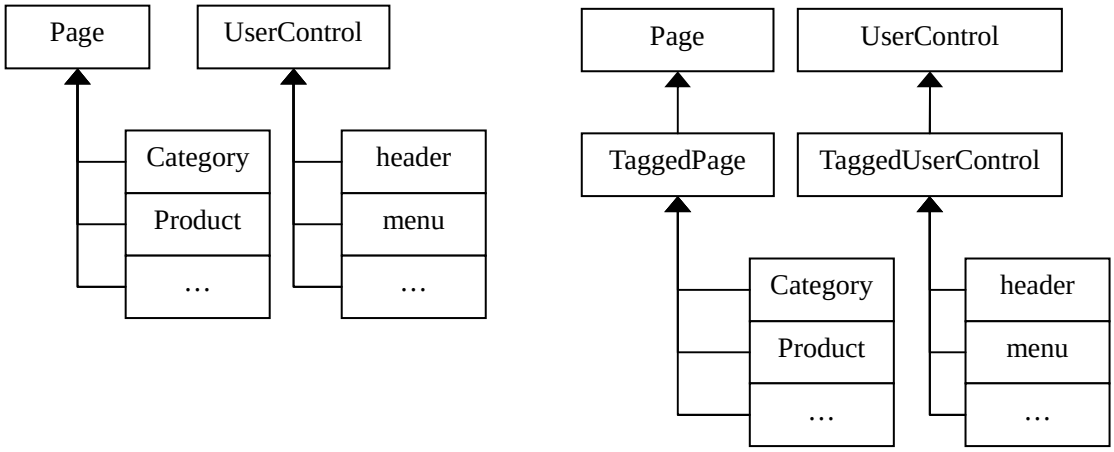
Note that the server is *not* required to skip the generation of the page or fragment even though its cache key is in the key list: it can optionally choose to include the actual output (with the `<cache>` tags). This flexibility allows a server to effectively remove the caching capability of an untrusted proxy, and to proactively update both the content and CVL when server deems necessary.

4.5 Summary of the protocol

The protocol can be summarized as follows:

- From server to proxy: uses `<cache>` tags to inform Proxy+ both the contents to be cached and the associated CVL tags. This allows on-demand and incremental installation of the CVL, and also affords server the opportunity to control the Proxy+ caching capability if necessary.
- From proxy to server: uses `<X-CachedKeys>` to inform already cached contents so that redundant computation and transfer maybe avoided.
- The advanced output cache feature VaryByCustom requires the server to specify an URL of a DLL that exports the function GetByCustomString which the proxy subsequently downloaded. This feature may not be possible if such function needs to access resources known only at the server side.
- Proxy+ is incrementally deployable: it behaves as any normal proxy with non-proxy+ aware applications (servers). The reverse is also true: no ill side-effect is caused when a proxy+ aware application interacts with a normal proxy.

The consistency control of Proxy+ cache is enforced exactly the same as the output cache on the server. Thus, our protocol accomplishes the goal of



a) class hierarchy before modification b) class hierarchy after modification

Figure 8. Web application presentation tier class hierarchy

replicating output cache functionality on the proxy. It should be noted that using DLL to package custom cache key generation function is only for our convenience of implementation on Windows platform. A complete implementation may provide equal support for other platforms by, e.g., using a cross-platform language like Java or script language.

5 Application Modifications

We now turn our attention to the necessary modifications to applications. As it turns out, the changes are very minor, and even trivial if supports are built into ASP.NET.

Proxy+ architecture currently targets ASP.NET applications and assumes programmers have used

```
bool cached;

override OnInit(e) {
      cached = (my cache key) ∈ (the key list received with the request);
                  // my cache key computed in the same way as on proxy
      base.OnInit(e);
}

override OnLoad(e) {
      if not cached
            base.OnLoad(e);
}

override Render(output) {
      if cached {    // make a placeholder for proxy to insert content at;
                        // still need the inner fragments to output their content
            output.Write (beginning of subst tag);
            foreach ctrl in this.Controls
                  if ctrl is TaggedUserControl
                        ctrl.RenderControl(output);
            output.Write (end of subst tag);
      } else {
            output.Write (beginnng of cache tag);
            base.Render(output);
            output.Write (end of cache tag);
      }
}
```

Figure 9. Pseudo code of the TaggedPage class

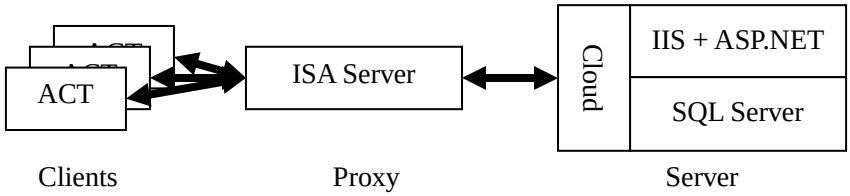


Figure 10. Experiment configuration

ASP.NET Web Forms Page and UserControl class to implement dynamic content caching on server side. To enable output tagging for such applications, it is sufficient to simply extend Page and UserControl class to generate tags and avoid regeneration of cached content. The process doesn't interfere with the original workflow of the application *at all*.

In general a Web application builds its UI by using ASP.NET Web Forms. The components at the presentation tier are subclasses of the class System.Web.UI.Page and System.Web.UI.UserControl as illustrated in Figure 8a). The content is generated by handling appropriate events in the class. For example, "Load" is a typical event which is handled by many pages and user controls to populate content to UI.

The modification to the application must satisfy two requirements. The new application should be able to recognize the list of keys sent from the proxy and avoid regeneration of corresponding content. It also needs to insert additional tags as described to enable the proxy to do fragment caching and page assembly.

We modify an application as follows. Two new classes TaggedPage and TaggedUserControl are added; they are subclass of Page and UserControl respectively. All subclasses of Page and UserControl will inherit from them instead, as shown in Figure 8b). These two classes override the event dispatching and HTML outputting functions in their superclasses. According to the key list attached in the request, they will decide whether the specific event need to be dispatched to the original handler or not (to avoid regeneration of cached content) and what additional tags (`<cache>` or `<subst>`) and CVLs are to be inserted into the HTML output.

The pseudo-code of TaggedPage is listed in Figure 9. TaggedUserControl's code is essentially the same with minor differences. For applications built with other Web programming platforms (e.g. JSP), we believe the modifications would also be similar.

6 Experimental Results

We measure the performance of our Proxy+ prototype with a representative e-commerce benchmark called .NET Pet Shop [12]. The availability of the source code allows us to experiment Proxy+ *without* hack into ASP.NET itself. The experiment configuration is depicted in Figure 10.

The clients run Microsoft Application Center Test (ACT) to simulate a number of concurrent Web browsers. ACT creates enough threads (specified by a connection number) to issue requests according to a test script that defines the test workload. The request distribution of the workload is shown in Table 2. The requests are sent to the proxy running Microsoft ISA Server with the output cache-enabled filter installed, and then forwarded to the backend Web server running Microsoft IIS, ASP.NET and SQL Server. The hardware settings of the proxy and the server are dual 1.7G Pentium 4 Xeon with 2GB RAM and dual 2.4G Pentium 4 Xeon with 1GB RAM respectively. The clients are also powerful enough not to become bottlenecks in our tests. All machines are connected in a switched 100Mbps Ethernet. A WAN emulator (Shunra\Cloud [18]) running on the backend server is used to set a constant latency between it and the proxy, while the client accesses the proxy over the LAN directly.

Activity	Percentage
Category Browsing	18%
Product Detail	16%
Search	18%
Home Page	18%
Shopping Cart	7%
Order	1%
Account/Authentication	22%

Table 2. Distribution of the test workload

We compare the response time of the Web site accessed via the proxy with and without the filter enabled (Proxy+ and common proxy). Figure 11 shows the average response time versus number of concurrent connections when the roundtrip network latency between the proxy and the server is set to 400ms.

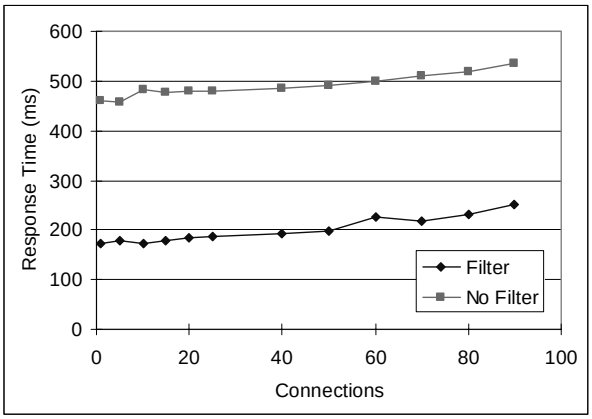


Figure 11. Response time with 400ms roundtrip latency

As can be seen, the response time of common proxy is always above the roundtrip latency because every request has to travel through the delayed link and so does its response. With Proxy+, about 60% of the response time can be saved, where the full page hit ratio (a page and all of its inner fragments hit the cache, thus avoiding server access) is about 70%.

Network traffic saving is another benefit besides response time improvement. In the above test, 87% of the traffic between the proxy and the server is reduced on average.

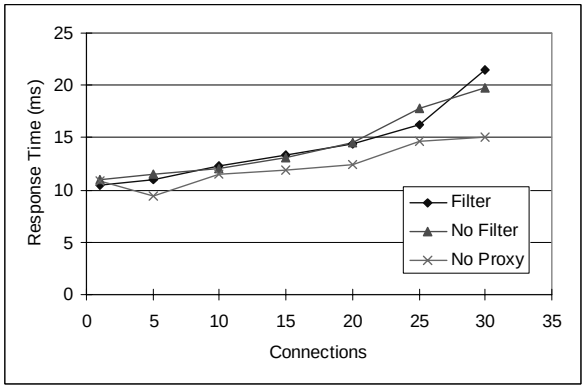


Figure 12. Response time without setting latency

To measure the overhead of the filter, we repeat the test without setting any network latency and also compare them with the response time of accessing the Web site directly. The results are shown in Figure 12, where “No Proxy” means requests are issued directly to the Web server. Due to ISA Server’s overhead, the response time of “No Filter” option is slightly more than that of “No Proxy”. And because the filter’s overhead happens to counteract the saved server side response time, “Filter” option has about the same response time as “No Filter”. Therefore, the overhead can be roughly estimated as the server side response time (about 10ms as shown by “No Proxy” data since network delay is negligible) timing the hit ratio, that is, about 7ms, which is acceptable when network latency between proxy and server is considerable, for example, hundreds of milliseconds.

7 Security aspect

Besides the security limitations that common HTTP caching proxies have, Proxy+ raises some different issues as well as interesting possibilities. Caching can be thought as filtering, consequently there is always a possibility of substituting contents. Proxy+ makes it possible for such actions – intentionally or otherwise, to be performed on a much finer granularity. We note that Proxy+ leaves the power and flexibility of control on the server. For a more systematic way to guard against proxy abuse we refer readers to approaches such as Gemini [15].

It poses a problem to proxies that they need to import DLLs provided by servers in order to cache pages/fragments with **VaryByCustom** attributes. Such DLLs must be signed by the server. If a proxy is unsure, it should run such DLL in a sandbox and deny its access to resources such as network and storage IO.

8 Conclusions

After establishing the argument that simple proxy extension will work just as well for dynamic content, we proposed Proxy+ architecture for dynamic content caching on augmented proxies near clients. The protocol uses simple extension to HTTP and can work coherently with common Web applications and proxies. Only minor modifications to existing applications are necessary to cooperate with Proxy+. Our experiment shows that a significant amount of response time and network traffic can be saved with Proxy+. Due to its low implementation cost and incremental deployability, we believe it is a competitive solution. We are currently extending Proxy+ fragment caching and page composition components from proxy servers to Web browsers so that clients with last-mile bottleneck can benefit from the bandwidth saving by this scheme [2][6][17].

Acknowledgements

We are very grateful to the anonymous reviewers for their helpful comments.

References

[1] BEA WebLogic Server. http://www.bea.com/products/weblogic/server/

[2] Brabrand, C., Møller, A., Olesen, S. and Schwartzbach, M.I. Language-Based Caching of Dynamically Generated HTML. World Wide Web 5(4): 305-323; Jan 2002

[3] Cao, P., Zhang, J. and Beach, K. Active Cache: Caching Dynamic Contents on the Web. In: Proc. of IFIP Intl. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), pp. 373-388.

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

[5] Datta, A., Dutta, K., Thomas, H., VanderMeer, D., Suresha and Ramamritham, K. Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, Madison, Wisconsin, USA, June, 2002, pp. 97-108.

[6] Douglis, F., Haro, A. and Rabinovich, M. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. USENIX Symposium on Internet Technologies and Systems, December 1997, pp 83-94.

[7] IBM WebSphere Application Server. http://www-3.ibm.com/software/webservers/appserv/

[8] IBM WebSphere Edge Server. http://www-3.ibm.com/software/webservers/edgeserver/

[9] Iyengar, A. and Challenger, J. Improving Web Server Performance by Caching Dynamic Data. In: Proc. of the USENIX 1997 Symposium on Internet Technologies and Systems (USTIS'97), Monterey, CA, December 1997.

[10] Labrinidis, A. and Roussopoulos, N. WebView Materialization. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Dallas, Texas, USA, May 2000, pp. 367-378.

[11] 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 (VLDB 2002), Hong Kong, China, 20-23 August, 2002.

[12] Microsoft .NET Pet Shop. http://www.gotdotnet.com/team/compare/petshop.aspx

[13] Microsoft ASP.NET. http://www.asp.net/

[14] Microsoft ISA Server. http://www.microsoft.com/ISAServer/

[15] Myers, A., Chuang, J., Hengartner, U., Xie, Y., Zhuang, W. and Zhang, H.. A Secure, Publisher-Centric Web Caching Infrastructure. Proceedings of Infocom '01

[16] Oracle9iAS. http://www.oracle.com/appserver/

[17] Rabinovich, M., Xiao, Z., Douglis, F. and Kalmanek, C. Moving Edge Side Includes to the Real Edge – the Clients. 4th USENIX Symposium on Internet Technologies and Systems, March 2003

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

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

[20] Yuan, C., Chen, Y. and Zhang, Z. Evaluation of Edge Caching/Offloading for Dynamic Content Delivery. In: The 12th Int'l World Wide Web Conference (WWW 2003), Budapest, Hungary, 2003.