Workshop on Computer Networks

An Efficient Design for Server-side Caching

Yeim-Kuan. Chang and Kun-Lin Chiang
Dept. of Information Management, Chung Hua University
{ykchang,klchiang }@mi.chu.edu.tw

Abstract

The World Wide Web (WWW) has been a very successful distributed system that can be used to share information with people on the Internet. The exponential growth of the Internet usage has led many new problems to be solved. Caching objects has been a very effective way to reduce the delay of repeatedly accessing web pages. However, server-side scripts generating web page on-the-fly are used extensively by web page designers in recent years. The web pages generated dynamically make the caching in client useless.

In this paper, an efficient design for server-side caching is proposed. The proposed design improves the existing server-side caching systems by using rewrite engine supported by apache web server and novel cache management. Simple rewrite rules run on the web server totally remove the overhead of validation process for checking the freshness of web pages. Cache manager that executes asynchronously removes the cache garbage collection from the critical path of accessing a web page. The results from the analysis and the performance measurements show that the proposed caching system has shorter response time and higher throughput than the traditional dynamic server-side scripts and the existing caching systems.

Keywords: Server-Side Caching, Dynamic web pages, URL rewrite, .

Contact Author :
Name : Kun-Lin Chiang
E-mail :klchiang@mi.chu.edu.tw
TEL : 03-5374281-6524
Postal Address : Dept. of Information Management,
Chung Hua University, 30 Tung Shing, Hsinchu, Taiwan, R.O.C.

1. Introduction

The World Wide Web (WWW) is an information distribution system based on a client/server model. The success of WWW has resulted in an exponential growth of the user population, the total host count, and the amount of total traffic on the Internet. We have seen that the hot pages create "hot spot" of the Internet. The hot spots on the Internet cause the corresponding web servers slow down or even crash. Upgrade of the servers does not solve the problem because the hot spots move around the Internet. We also have seen that the same pages have been transmitted over the same network links again and again to thousands of different users. Caching can be very effective at reducing network bandwidth consumption as well as balancing servers' load [Gwer94, Ding96, Abra95, Liuc97].

Tim Berners-Lee has made a proposal that calls for research in replication [Bern95]. He thinks that caching without replication will not be adequate for distributing popular web pages. Often users request an object many times in a short period of time as they surf on the Internet. Also, the pages accessed by a user may be accessed again by other users from different machines. Thus, caching WWW pages can reduce the access latency, network bandwidth consumption, and web server load. Caching also provides a method to analyze an organization's access patterns.

The above description for caching is perfectly suitable to the web pages statically stored on disk. Figure 1 shows that the web server reads the content of a web page from a disk file and send the content back to the client. Unfortunately, caching in the WWW today has been seriously limited by the non-cacheability nature [Doug97,Cao98] of the ever-increasing dynamic contents, including data computed by CGI programs, documents queried from database management systems, and pages generated by typical web applications written in Microsoft ASP or Unix PHP. Figure 2 illustrates the data diagram of a request for the dynamic web page generated by a server-side script. Web server asks PHP engine to load the user's script from disk, compile it, and execute it to generate the output web page. This process is much slower than that of directly reading the content in a file from disk. We will show the performance comparison between these two later.

Caching is impeded furthermore by the cache-busting techniques for increasing the advertising revenues. Content providers use cache-busting techniques to vary the advertising banners or other information that appears on a page, every time the page is accessed. One of the purposes for the cache-busting is to count the hits on the servers of content providers, called *hit-metering*.

In addition to the uncacheability of dynamic web pages, the server-side scripts must be executed on the web server every time they are requested. It is the main reason that the web server can be overloaded by only a small number of requests. This in turn increases the access delay of clients. As the network bandwidth increases in a very fast rate, how to reduce the server's overloading problem becomes very important.

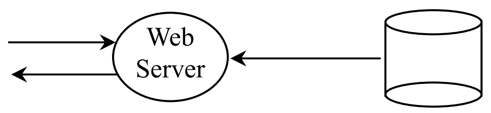

Figure 1: The flow diagram of requesting a static page.

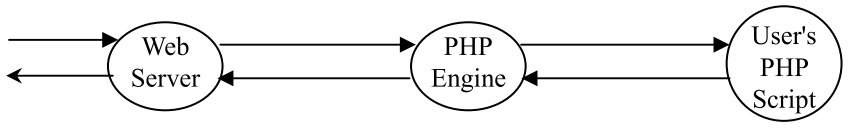

Figure 2: The flow diagram of request for a page generated dynamically by a PHP script.

Although the web pages generated by server-side scripts are called "dynamic", they may not change in every second. A lot of dynamic web pages are intrinsically static, not changed even in a longer period of time. For example, the official results of the National Science Foundation project approvals will be posted unchanged on its web site for at least three months. However, the results are dynamically queried from databases using the ASP scripts. Therefore, for these kind of web pages, the simplest approach is to dump the content generated by the server-side scripts to the server-side's disk. Then the page generation process is eliminated as long as we can make sure that the cached content is up-to- date. Additional validators, such as the content length, last modification time, entity tags, and cache control [HTTP1.1], are also needed in order to make them cacheable in the proxy servers.

In this paper, we propose a server-side caching system that can be used by web page designers to store the cacheable dynamic web pages on disks. It is not assumed that we know the precise expiration times of web pages in advance. A heuristic validation model is used. The rewrite engine [rewrite] is utilized in the proposed caching system. Rewrite rules make the process of validating freshness of web pages fast. We compare the proposed system with the well-know caching system phpCache [phpC] and the traditional no-cache system and show that the proposed system is superior.

The rest of the paper is organized as follows. Section 2 describes the phpCache in detail and shows it weakness. The proposed caching system is described in Section 3. We will give a comparative study for the traditional plain text, the server-side scripts with no cache, phpCache and the proposed system in Section 4. The related works are described in Section 5. Finally, a conclusion and future work are given in the last section.

2. Existing caching systems

The phpCache is an application-level caching system that can cache the web pages dynamically generated by PHP scripts [PHP]. In addition to generating the output web page and sending it to the client, phpCache also stores the output web page in a file on disk, a.k.a. the cache, if the file does not exist. Thus, the subsequent request for the same PHP script can be satisfied by the file in cache without executing the same PHP script again. All the cache management tasks including garbage collection are done in the level of PHP scripts. Functions Cache() and Endcache() in phpCache are the interfaces that need to be familiar with for script programmers. Figure 3 shows the flow diagram of phpCache. All the requests for PHP scripts are passed to PHP engine. The engine first calls the function cache() to check if the requested web page has already been stored in cache freshly. If the web page exists and is fresh, the content of the web page is directly returned to the client via the PHP engine. The function cache() is slow because of the following drawbacks. Firstly, The meta data of the output web page is stored in the file along with the content of the page. Therefore, it is slow to read the information from the disk file to check if the output web page is fresh or not. Secondly, cache() also performs the tasks of garbage collections.

If the web page does not exist in the cache or is not fresh, the user's PHP code is executed and the output web page is stored in cache by the function Endcache(). Finally, the output web page is returned to the client.

Another PHP-based caching system is jpCache. It improves over phpCache by introducing the 304 response, compressing the content, caching both GET and POST requests, and storing the objects in files or MySQL database.

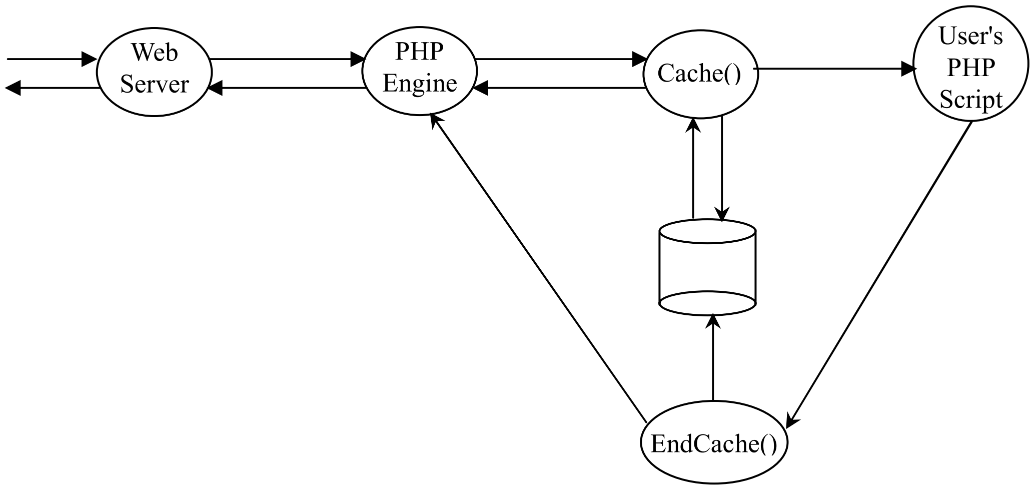

Figure 3: The flow diagram of the request for a page generated dynamically using phpCache caching system.

3. Proposed caching system

As described above, the process of accessing a static file is much faster than that of accessing a dynamic web page. Therefore, the basic idea of our proposal is to store the frequently accessed dynamic pages as static files in cache, similar to the phpCache. In addition, the overhead for validating the freshness of the cached page is minimized. That is, we like to make the delay of accessing a cached dynamic page as close to that of accessing a purely static page as possible. We achieve this performance improvement by moving the process of validating the freshness of a cached web page from application level to the level of web server, by using the rewrite module. This way reduces the overhead of excess buffer copying from web server to the application. To be specific, we design the proposed caching system in such a way that when a cached page exists in the system, it must be fresh. The out-of-date pages will be removed from the cache by the cache manager asynchronously. In other words, the garbage collection performed in background asynchronously is removed from the critical path of accessing the dynamic pages. The data flow diagram is shown in Figure 4.

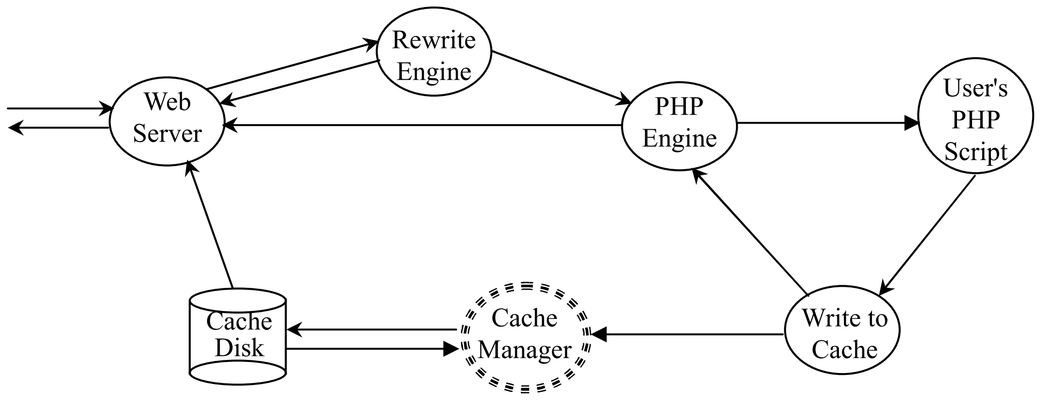

Figure 4: The flow diagram of requesting a PHP page using proposed cache system, reCache.

The format of URLs released to the public from the proposed caching system is the type B. Type A format is only used internally in the system. A rewrite condition rule using PATHNAME variable can be used to check if the cached file corresponding to the requested web page exists in cache. Therefore, upon receiving the request for a fresh web page, the web server passes the request to rewrite engine that in turn passes "exist" message back. Then the web server can directly read the content of the file and returns it to the client. Since the rewrite engine is always compiled inside of web server as apache, the rewrite rule executes very fast. Thus, the time taken for accessing the fresh web page is almost the same as that of accessing a static web page with the same page size.

If a request is for a web page that does not exist in cache, the web server will pass the request to the PHP engine for executing the user's script. The user's script is programmed in such a way that the output page generated dynamically is written to a disk file using type B format. The PHP engine in turn informs the web server to send an internal sub-request for the newly created file. Finally, the cache manager is also informed about the addition of the file by a message containing a newly created URL from Write-to-Cache module.

Since the expiration time of the output generated by a server-side program can not be predicted accurately, we employ a weakly consistent model (as in Squid) to validate the dynamic web page. In the weakly consistent model, it is assumed that the more (less) frequently a page is changed in the past, the more (less) possible that it will be changed in the near future. The cache manager is responsible for carrying out the cache consistency model. The cache manager maintains the meta data of the cached file in a table shown in Table 2. The URL is in format of type B. LMT means last modify time.

URL	MD5 (P)	Expiration time	LMT
URL1	MD5(P1)	Exptime1	LMT1
URL2	MD5(P2)	Exptime2	LMT2
URL3	MD5(P3)	Exptime3	LMT3
…	…	…	…

Table 1: an example of meta data kept in cache manager.

The cache manager runs asynchronously. In other words, it is activated at time Exptime1. For every page P whose expiration time is equal to Exptime, cache manager does the following:

```
1. Update the entries received from Write-to-Cache module by computing the
   associated MD5 and LMT, and set the expiration time as current_time +
   default_expire. The default_expire can be determined by webmaster using the
   change characteristics of the pages,
2. remove the record of URL in the table,
3. rewrite the URL into type A format and generate the page dynamically as NewP,
4. compute md5(NewP)
     if md5(NewP) != md5(P) then
         a. insert (URL, md5(NewP), current_time+default_expire) to table,
         b. save page NewP to file, (LMT of the file is changed to current time)
     else
        a. insert (URL, md5(P), (expiration_time - LMT)*1.1+LMT) to table,
        (LMT of file is not changed)
```

Notice that the 1.1 above is the parameter of weak consistency model which can be determined by web master. As stated earlier, one unique property of the proposed caching system is that if the requested file exists, it must be fresh. We achieve this by the rewrite rule whose algorithm is listed as follows:

```
if the requested file exists, returns the file or 304 if the request is IMS
else
    1. rewrite the requested URL and execute a program to generate the page
       dynamically as NewP,
    2. allocate a space of size(NewP) by performing the replacement algorithm
       replace(size(NewP)),
    3. save page NewP to file
    4. insert (NewP, md5(NewP), LMT(file)+default_expire) to table,
    5. return NewP to client
```

Figure 5 shows an example of a rewrite rule that checks the requested URL (denoted as REQUEST_FILENAME). If the cached file corresponding to the requested URL does not exist in the cache, the rewrite rule then decides to load the corresponding server-side script from disk, compile it, and run it. As we can see the else-part of above algorithm is implemented in this server-side script. The function replace(size) performs the replacement policy of the cache. We uses the LRU. Various replacement policies can be used such as largest size first, popularity-aware algorithms, etc. Frequency-based [Jin99, Arli99] and cost-aware [Cao97] algorithms are the most recent work proposed in the literature.

```
RewriteMap    urlparse   prg: /urlparse.pl
RewriteCond   %{REQUEST_FILENAME}     !-s
RewriteRule   ^/(.+)/(.*)\.html$      $1${urlparse2:$2}      [L]
```

Figure 5: a sample apache rewrite rule.

In this paper, the format of URLs is classified into two types as shown in Table 2. Type A is the traditional URL format with the query string when the client requests a dynamic web page using the GET method. Since type A URL contains a question mark (?), the client side cache usually does not cache this page. In order to remove the question mark in URL and allow the client-side caches the dynamic page, we define the corresponding type B format. By using a simple rewrite rule running on the web server, the web page generated dynamically is possible to be cached as long as the relevant HTTP headers are present. As we can see, embedding the pairs of keyword and value in URL using GET method loses the flexibility of users inputs. This is where POST method comes from. To imitate the actions of POST method, we allow users input the keywords and values but still using type B format. This is done by a simple javascript code. Figure 6 shows the code of a sample program with two pairs of keywords and values.

```
Type A URL Format:
    http://site/example.php?k1=v1&k2=v2
Type B URL Format:
    http://site/example.php-k1=v1_k2=v2.html
```

Table 2：URL formats

```
<SCRIPT LANGUAGE=JavaScript>
function Location() {
   var UrlStr;
   UrlStr = "http://site/example.php- k1=" + document.LocationBody.Key1.value +
     "_k2=" + document.LocationBody.Value1.value + ".html";
   window.location.href = UrlStr;
}
</SCRIPT>
<FORM NAME=LocationBody>
    Key 1: <INPUT TYPE=TEXT NAME=k1><BR>
    Key 2: <INPUT TYPE=TEXT NAME=k2><BR>
    <INPUT TYPE=BUTTON VALUE=Submit onClick="Location();">
</FORM>
```

Figure 6: A sample javascript code to imitate the action of POST method.

4. Performance Evaluation

We have implemented the proposed caching system using PHP scripts as examples. A number of experiments are conducted to show the performance improvement of the proposed system over others. The httperf performance tool for web servers is used. Average latency and server throughput are the two metrics used in the comparisons. Table 3 shows the parameters of software and hardware in the experiments.

Hardware		**Software**	
CPU	PIII 450	OS	RedHat Linux 7.2
RAM	448 MB	Web Server	Apache 1.3.26
HDD	Maxtor 10G 7200rpm	PHP	PHP 4.2.0
NIC	Intel 82559	phpCache	v1.4
		Measurement Tool	httperf-0.8

httperf parameters			
client	0/1	recv-buffer	16384
server	localhost	num-conns	10
port	80	num-call	1
send-buffer	4096		

Table 3：experiment parameters.

Notice that we are interested in the performance overhead incurred from the proposed caching system, phpCache, the traditional PHP system. Thus, we only measure the performance when the system load is light. To fully understand these overheads, we first conduct the measurement for a very simple "hello world" example. From this example, almost all the delays come from the caching system itself. The experiments Plain shown in the figures mean accessing static pages containing only two words “hello” and “world”. We denote our proposed caching system as reCache which means a cache system using rewrite. Plots for PHP in the figures represent the experiments for traditional PHP programs without any caching supports.

Figure 5 shows the performance of the requests for the web pages when cache does not contain a fresh copy (First Access, FA). Figure 6 is for the case when cache contains a fresh copy of the requested web pages (After First Access, AFA). We can see that phpCache performs much worse than other systems. This is because the overhead of phpCache comes from functions Cache() and EndCache(). The performance of the proposed system is very close to that of accessing plain files with the same sizes. The results of our experiments do not conflict with the results given in [phpCache web site]. In [phpCache web site], the user’s PHP programs contain various degree of computation for generating the output web pages. However, the hello word example only contains a simple statement “echo hello world”.

Figures 7 and 8 illustrate the results of the similar experiments for different page sizes. We can see that the performance of accessing plain text is almost the same as that of reCache AFA with fresh copies of web pages in the cache. The performance of reCache FA is similar to that of traditional PHP programs. The performance improvement for reCache AFA is eight times over phpCache AFA for the cached pages. Similarly, the performance of reCache FA is two times over phpCache FA for the pages not stored in the cache.

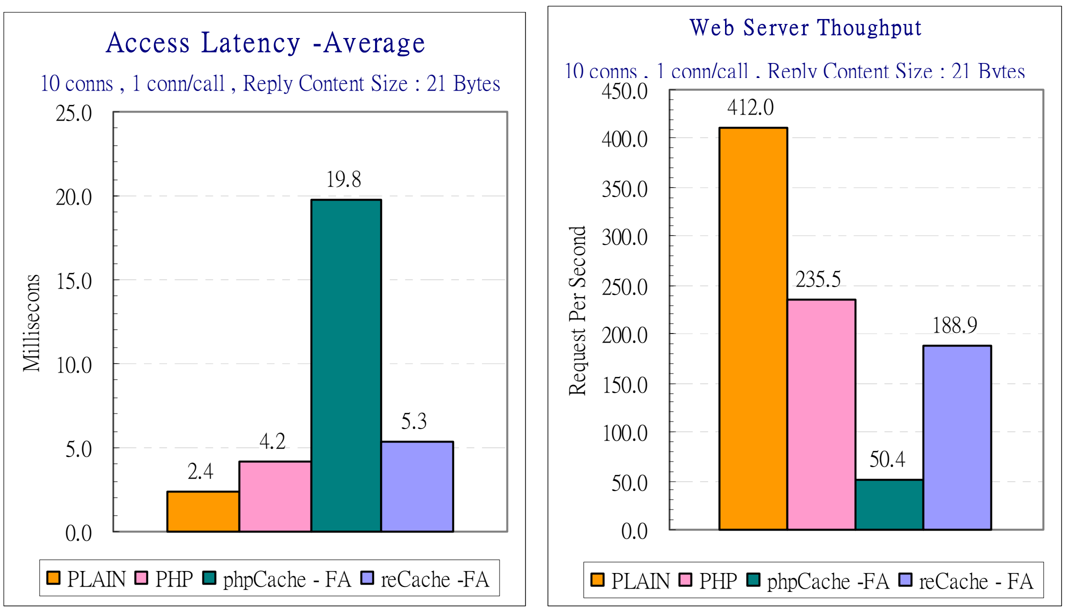

Figure 5: The average latency and server throughput of a request for a hello world example before it is cached.

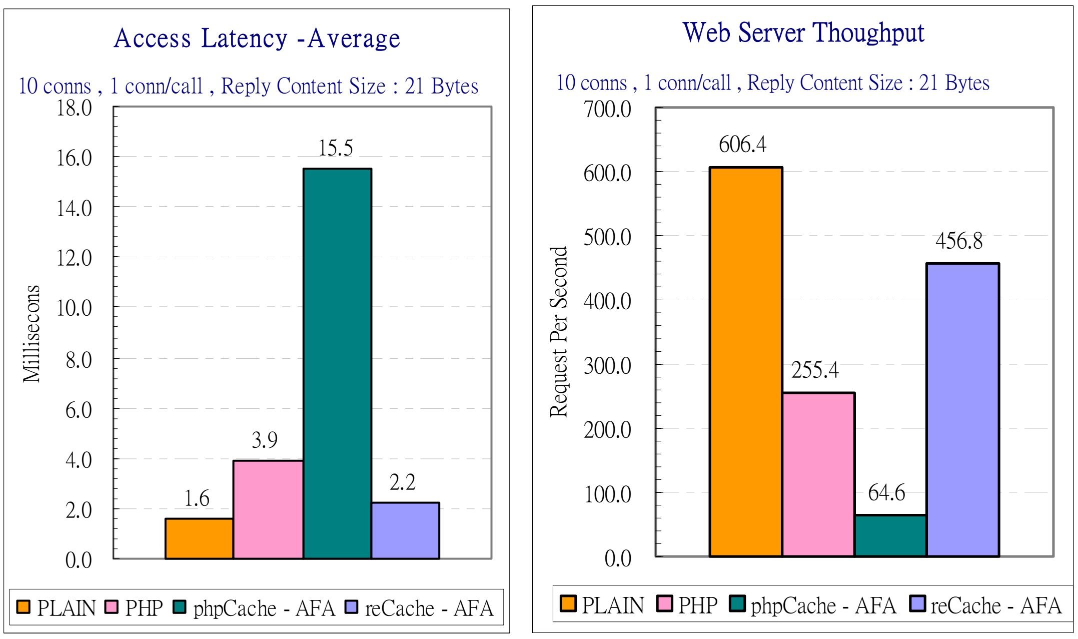

Figure 6: The average latency and server throughput of a request for a hello world example after it is cached.

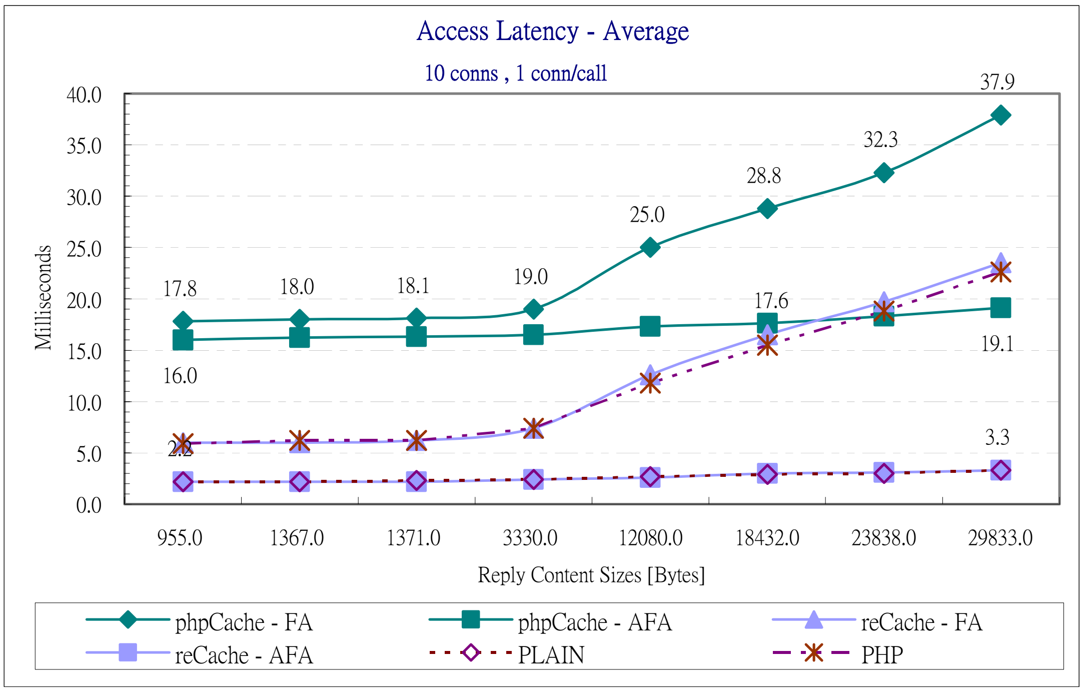

Figure 7: The average latency of requests for dynamic web pages of different sizes.

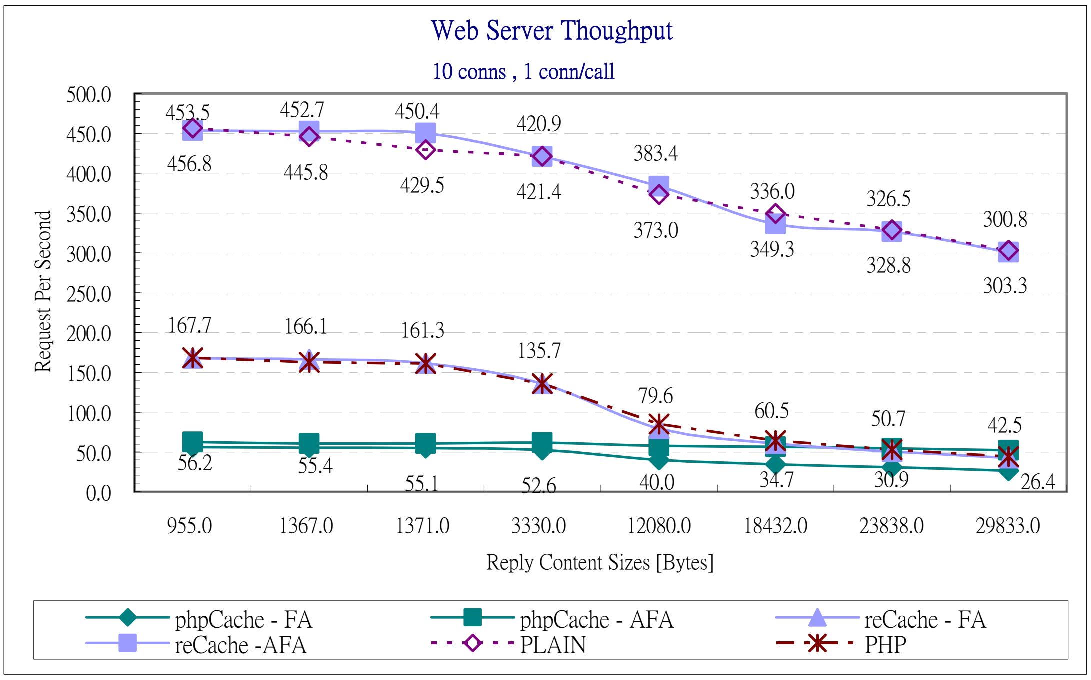

Figure 8: The average server throughput of requests for dynamic web pages of different sizes.

5. Related Work

There have been many studies to examine the cacheability problem from various perspectives. Many approaches were proposed to improve performance of accesses to the dynamic web contents. Douglis et. al. presented an HTML Pre-Processing extension, called HPP, to support resources that contain static and dynamic elements. HPP takes an application-specific approach to separate the static and dynamic portions of a resource. The static part of a resource called template can be cached and the dynamic part of the resource is retrieved on each access. They suggested using Java applets to implement the proposed protocol for avoiding modification of client software. However, if java applets are used, the rich set of HTML tags will lose their role and the rendering capability of web page will be limited. The task of formatting pages will be the responsibility of the programmers of the applets.

In [Cao98], Active Cache moves parts of server processing on each request to the caching proxy in a flexible, on-demand way via so-called cache applets. The idea of caching static documents is extended to cache objects, i.e., data with a method that is invoked when data are supplied from caches. The cache applets can deposit information in a log object, which is sent back to the server periodically and when the applet or the document is purged from the cache. The major disadvantages of Active Cache are as follows. The size of cache applets may be too large (~ 1-3KB) compared to that of average size of original documents (~ 10-30KB). This accounts for 10% cache space overhead. Thus, cache applets can only be used for certain selected resources. The response time is also increased if the the requests go through cache applets, as stated in the paper. To implement the proposed Active Cache protocol, the web server and the proxy server must be modified, which may lower the acceptance rate.

The WebExpress system utilizes the repetitive and predictable nature of transaction processing to enable the caching and differentiating functions to work with minimal data transfer. WebExpress allows the content provider to provide a base version of a dynamic resource and only send delta-encodings (differences) against the current version. In [Mog] also showed that delta-encoding can provide remarkable improvements in response-size and response-delay for an important subset of HTTP content types. TopBlend proposed in [Top] describes a new HTML differencing tool implemented in Java. TopBlend can be employed easily by the website tracking services.

6. Conclusions

In this paper, we have proposed a novel approach for caching the output pages generated dynamically from server-side scripts. The proposed system utilizes the rewrite engine to avoid validating the freshness of cached web pages. The proposed protocol also eliminates the need of executing garbage collection at the time of the request arrival. The garbage collection process is run in background asynchronously. The analysis and the performance measurements show that the proposed caching system outperforms the existing systems.

We believe that our work in this paper is easier for web pages designers to understand and implement in their server-side scripts. Our work is the first step to make the dynamic web pages have the similar access delays to the static pages. As our future work, we will first characterize the web resources by analyzing the web resources recorded in the access log file. The aim of characterizing web resources is to better understand how often the resources change and how the changes in the content and metadata of the web resources affect caching by browsers and proxy servers. This part of research involves improving the cacheability of resources like advertisement links and banners, and distinguishing between static and dynamic portions of a resource generated by typical web applications. Intuitively, processing non-cacheable resources in proxy servers in fact adds more loads on proxy servers and further delays the data transmission. Thus if we can predict which resources can be cached and which ones can not, we can lessen the load of proxy servers by requesting the non-cacheable documents directly from original servers and thus reduce the access delays. We will develop an algorithm to determine which web resources are cacheable before making the requests.

As we can see, our work in this paper does no good for the pages whose content changes all the time. We will also study the possibility to separate the dynamic web pages into two parts, the static part and dynamic part. For example, the stock share quotes are dynamically changed every second. Every time we click on the link of a stock share index, the whole page is transmitted over the network to the client. Let's assume the stock share quotes are changed every five minutes, the time when the proposed algorithm can be effective is also only five minutes. Can we make the cacheability of these kind of web pages better? By carefully looking into the web page of the share price and its HTML source code, we can see that only a small portion of the values in the page are changed.

.

References

1. [Abra95] Abrams, M., Standridge C. R., etc., "Caching Proxies: Limitations and Potentials", In Proceedings of the Fourth International World Wide Web Conference, December 1995, http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html.
2. [Arli99] M. Arlitt, et. al., "Evaluating Content Management Techniques for Web Proxy Servers", In Proceedings of the 2nd Workshop on Internet Server Performance, May, 1999.
3. [Bern95] Berners-Lee, T., "Propagation, Replication, and Caching on the Web", 1995, http://www.w3.org/pub/WWW/Propagation/Activity.html.
4. [Cao97] Pei Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", Usenix Symposium on Internet Technlogy and Systems (USITS-97), 1997.
5. [Cao98] P. Cao, Jin Zhang and Kevin Beach, "Active Cache: Caching Dynamic Contents (Objects) on the Web", In Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, The Lake District, England , September 1998.
6. [Ding96] Dingle, A. and Partl, T., "Web Cache Coherence", In Proceedings of the Fifth International World Wide Web Conference, Paris, France, May 6-10, 1996, http://www5conf.inria.fr/fich_html/papers/P2/Overview.html.
7. [Gwer94] Gwertzman, J. and Seltzer, M., , "The case for geographical pushing-caching", Technical Report HU TR-34-94, Harvard University, DAS, Cambridge, MA, 1994.
8. [HTTP1.1] Fielding, R. et. al., "Hypertext Transfer Protocol - HTTP/1.1", HTTP Working Group, Internet-Draft, draft-ietf-http-v11-spec-rev-03, March 13, 1998.
9. [Jin99] S. Jin and Azer Bestavros. Popularity-aware greedy dual-size algorithms for Web access. In Proceedings of the 20th Int'l Conf. on Distributed Computing Systems (ICDCS), April 2000.
10. [Liuc97] Liu, C. and Cao, P., "Maintaining Strong Cache Consistency in the World-Wide Web", In Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, May 1997.
11. [Mog] Jeffrey C. Mogul et. al., "Potential benefits of delta-encoding and data compression for HTTP", In Proceedings of the ACM SIGCOMM '97 Conference, September 1997.
12. [PHP] http://www.php.net
13. [phpC] http://0x00.org/php/phpCache/
14. [rewrite] Apache, "Apache HTTP Server Version 1.3: Module mod_rewrite URL Rewriting Engine"
15. [Top] Yih-Farn Chen et. al., "TopBlend: An Efficient Implementation of HtmlDiff in Java", AT&T Labs-Research Florham Park, NJ USA , 2000