Journal of Physical Sciences, Vol. 14, 2010, 181-189
ISSN: 0972-8791, www.vidyasagar.ac.in/journal
Published on December 24, 2010

A Dynamic Web Caching Technique for using "URL Rewriting"

Amarjeet Singh[1] and Rakesh Ranjan[2]

[1]Department of Computer Applications, Institute of Environment and Management, Lucknow, Uttar Pradesh, INDIA
E-mail: amarjeetsingh_9@rediffmail.com

[2]Department of Computer Applications, Institute of Environment and Management, Lucknow, Uttar Pradesh, INDIA
E-mail: rakeshranjan.lko@gmail.com

Received September 8, 2010; accepted November 1, 2010

ABSTRACT

In order to make surfing the internet faster, and to save redundant processing load with each request for the same web page, many caching techniques have been developed to reduce latency of retrieving data on World Wide Web. In this paper we will give a quick overview of existing web caching techniques used for dynamic web pages then we will introduce a design and implementation model that take advantage of "URL Rewriting" feature in some popular web servers, e.g. Apache, to provide an effective approach of caching dynamic web pages.

1. Introduction

Two decades ago, most of web pages on the internet were just static HTML pages and the job of building a website was an easy job and doesn't require much knowledge of computer programming. But nowadays most of websites are dynamic ones which rely on a scripting language to run, and become more complicated as they provide new features such as personalization of contents and dynamic advertisements. Also it allows just about anyone, with limited or no web design experience, to update their own website via an administrative backend. But unfortunately loading of a web dynamic page is slower and requires more operations than loading a static page. As a result the need of web caching becomes crucial as dynamic pages are the dominant in the web. To alleviate server processing overhead and reduce loading time of dynamic web page, it is possible to get the generated output of the dynamic page and save it in the cache, so that any subsequent request will be served from the cache only with no need to execute the same code again. Thus a dynamic web page can be served nearly as a static one.

In this paper we will discuss the popular techniques of web caching used today, and then we will introduce a simple design model and its implementation of a web caching technique followed by a comparison study and an analysis of the enhanced performance after using this technique.

2. Web Caching Overview

Web caching can be subdivided into client, network and server caching.

2.1 Network Caching

The main end points of web page path are the origin server which created the response and the client which receives the response. Along the way there are other nodes such as ISP servers and proxy servers. For these servers' the problem of saving the bandwidth is an important one, and if the same page has been requested more than one time and gave the same response, it should be cached at the proxy server so that any client requests the same page, the proxy doesn't have to waste additional bandwidth in contacting the origin server and getting the page once again. This type of caching is called network cache, which mainly aim to save network traffic and bandwidth. In this type of caching the user-perceived response time should be lesser than if the response has to travel all the way from the origin server to the client, though loading time in network caching should be more than client-side caching, as in client-side responses are stored in the client's local machine. Many systems were developed to achieve network caching; examples of such systems are Squid proxy and Active Cache.

Network caching is subject to the drawbacks discussed in the previous section, and the same techniques are used to reduce these drawbacks.

2.2 Client-Side Caching

Client-side caching is mainly integrated into the user browser, in which the browser assigns certain disk space to store some latest pages the user has visited. The main goal of client-side caching is to reduce the request time for the client's web requests. Yet it may involve some drawbacks when the user's cached pages become stale, in order to solve this problem HTTP standards have some headers that help to reduce the probability of caching stale pages. The "Cache-Control" header controls the page cache ability, and it is a general header used to specify directives that must be obeyed by all caches along the request, "Expires" header specifies the expiration time of the page at which the page will become stale and has to be fetched again from the original server, "Last-Modified" header specifies last modification of the requested web page, the browser then compares the date of its cached version of the page to the server response, if it was saved before the "Last-Modified" header then it will fetch the page from the original server.

Client-Side caching is very effective for static pages, but in case of dynamic pages, caching headers are not effective as they depend on last modifications of the file on the file system, and since dynamic web pages are generated dynamically, so

script file responsible for generating the output doesn't necessarily change in order to give different output.

2.3 Server-Side Caching

As the name implies, server-side caching takes place at the origin server. The main goal of server side caching to achieve is to alleviate server load and to save processing of redundant requests. Server load can be specified into two categories, scripting engine load, and database server load. Many techniques have been developed to save server processing load, one of these techniques is Memcached which mainly concerns of alleviating database load. Memcached is a distributed memory object caching system, which caches database query results in memory. Another technique of database caching is content aware caching (CAC), which is distributed model for database caching that caches the results of database queries on the edge server. DBCache and DBProxy are example systems of CAC.

An alternative approach to CAC is Content-Blind Caching (CBC), in which edge servers store remote database query results independently. By version 4.0.1 MySQL represented "MySQL Query Cache" as a caching feature in its database engine, MySQL official documentation says "The query cache stores the text of a SELECT statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again. The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client".

In our implementation we will show how our model can save database and scripting load at original server and ensures the consistency of the contents of pages. We chose MySQL query cache to make a comparison with our model to depict the difference in performance, as it is a server-side caching as our model, and for its popular use on the internet.

3. Security Considerations

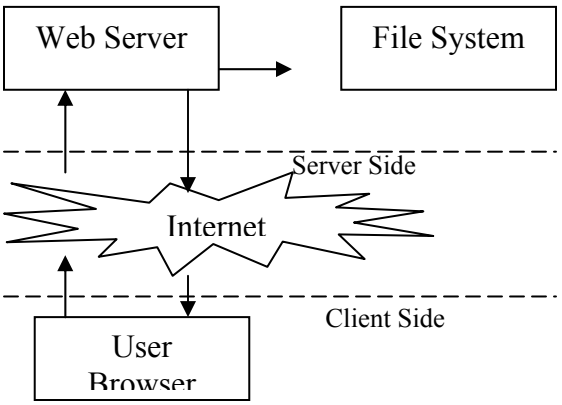
Although we do not focus on the security issues of web caching in this research, but it worth to mention some security considerations. W3C says: "Caches expose additional potential vulnerabilities, since the contents of the cache represent an attractive target for malicious exploitation. Because cache contents persist after an HTTP request is complete, an attack on the cache can reveal information long after a user believes that the information has been removed from the network. Therefore, cache contents should be protected as sensitive information".

4. Affect Loading Time at Server

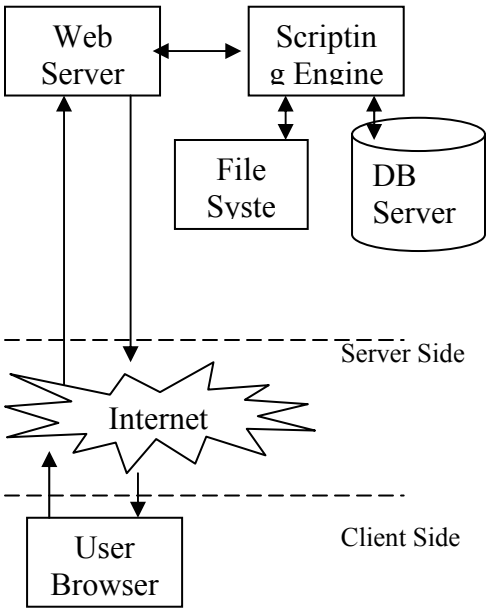
A static web page is a web page that always comprises the same information in response to all download requests from all users, dynamic web page is different from static web page, in which different output may be given according to other factor, such as user requesting the page or data or request type, etc…, so that serving a dynamic Web page involves processing overhead compared to serving a static one.

In static web pages the request and response process is simple, when a user requests the page, the web server receives the request and gets the page file from the file system, read it and output the contents to the user directly. We can note here that in order to process the page, two layers are involved, the web server and the file system. Whereas in dynamic web pages the process is much complicated, Fig. below shows a dynamic web page generation, we can see that when the web server receives the request it will check the requested page extension and compare it to its preconfigured list of extensions of dynamic pages, when the web server finds out that the requested page is a dynamic one, it will load the associated scripting engine such as PHP or ASP (Active Server Pages) with the page name as a parameter, which will read the page file from the file system and executes the code inside the dynamic web page file, and if the web page requires data from the database, then a connection should be established to the database and a query should be executed, then the database answers the query results back to the scripting engine and after code interpreting is completed, the scripting engine gives the output to the web server, which in turn gives it back to the user. So in dynamic web page execution model two more layers are added over those for static page, which are the scripting engine and database server, and that is done in the most cases. However, in other cases that process can go longer; consider a web site communicating with web services on remote server, the connection latency and web service execution time will be added too.

In order to clarify how much delay and increase in loading time occur when more layers are added to the process of retrieving a web page, we will run four test cases to demonstrate this point. In our comparison we will store 1024 characters in the subject webpage, which will run under Apache 2.2 web server and the dynamic version of the webpage will be written in PHP, and we will use PHP5 as scripting engine and MySQL 5 as the database server, the host computer is Pentium 4 2.0Ghz with 768MB of RAM. The resulting loading time will be the average loading time of 100 consecutive requests for the page.



Request/Response Process of Static Web Page



Request/Response Process of Dynamic Web Page

The first test case has the 1024 bytes of data stored in a plain HTML file, the second case has the data stored in a PHP file, and we will write the 1KB data inside the file and not within PHP tags, so the PHP interpreter will be loaded but should not interpret any code, whereas in the third case we will get the PHP interpreter involved, by printing the data using 'print' function, and in the fourth case we will retrieve the same data from a database table and use the PHP 'print' function to output the data.

5. Results

Table 1 shows that static page loading is the fastest of all as it involves only web server management to get the contents to the user. While in the second case, the PHP interpreter has been given the page to interpret, although there was no code to interpret, calling the PHP interpreter or scripting engine in general caused the extra load in serving the request. In the third case, the PHP interpreter executes the code that output the contents. So the loading here is web server management + scripting language engine loading + executing code. As the results show, there was not a big difference in loading time between this case and the second one, since loading the scripting engine is mostly more time consuming than code interpretation.

In the last case, a third-party was involved, which is the database management server, in our case 'MySQL' , so the loading time will be = web server

management + scripting language engine loading + executing code + MySQL processing of data. As we see the more dynamic the page becomes, the more latency we will get in order to get the output**.**

Table 1: Result Comparison

Case	Time in Milliseconds
1	4.2
2	7.24
3	32

6. Implementation

As we examined, static pages are much faster than dynamic ones as they do not involve any interpreter engine or database server. We noted that for each request of the same page, same operations are done every time which leads to the same output. Our goal is to save server load of subsequent requests by saving a copy of the output on the server, so that for any subsequent requests for the page, the server will treat that copy as a fully static page and will not load any third-party applications since the original data has not been changed.

6.1 Writing to the Cache

The process of writing to the cache will be as follows.

When the user requests a dynamic web page for the first time, the web page will be executed, any code inside the page and any database queries will be executed normally, and at the end of the execution, the script should save the HTML output as .html file in the cache directory. Saving the HTML output can be done by output control functions, such as ob_get_contents() in PHP.

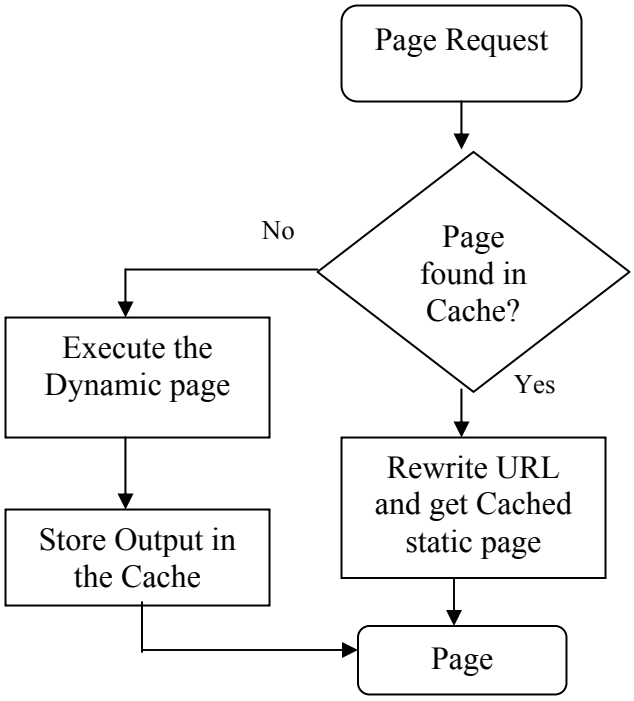
If any changes occurred to a database table, all cached pages that read from this table has to be deleted, and that can be done by implementing a class in the desired server-side language, that bind each page with its associated database tables, and when an alternation to the table contents takes place, that class - which have a reference to all associated cached pages – can delete them, then when a user requests the page, the dynamic version will be executed again for one time and will save the output as .html file in caching directory. The cached file name should have the same name of the dynamic page concatenated to its parameter, as the same dynamic page can give different output according to the passed parameters in the URL. For example if the user requests “http://example.com/profile.php?uid=saleh”, the cached HTML file name could be “profile.php?uid=saleh.html” or any other convention the web developer may follow.

The web site backend or administrative pages that add, edit or delete from the database should consider the cached pages that interact with the altered tables by cleaning the cache from pages that no longer have fresh contents.

6.2 Retrieving Cached Pages

When a second request to the same dynamic page comes to the server, the server will look for the cached file name in the cache directory, in our case "profile.php?uid=saleh.html", and if the file exists in the cache, the server will implicitly redirected to the cached page, and if the server didn't find it, it will let the request to call the original page with the sent parameters.

The decision of redirection should not be made by another dynamic web page; as if it is, the script engine will be loaded and then we didn't save script engine processing load. We want the web server itself to do the check and redirection, so the solution is "URL Rewriting" feature. "URL rewriting" or "Rewrite Engine" is a feature or module embedded in many web servers, that modifies the URL appearance or redirect the requested URL to a new one whether explicitly or implicitly, and used mainly to provide short and neat URLs. Rewrite engine is integrated in many web servers such as Apache and IIS (Internet Information Services). The flow chart of the model is shown in Figure below. The job of web server in managing cache is to check for cached page existence and then take a decision either to redirect to cached page or to call original dynamic page. In Apache, for example, URL rewriting module has two very useful options to accomplish this task, "RewriteCond" which specifies conditions according to which the next directive will operate, and this directive is the responsible of checking the existence of a cached copy in cache directory, and "RewriteRule" which will be executed if the "RewriteCond" is satisfied, which will perform the implicit redirection to the cached page if found in cache.



Flow Chart of Caching using URL Rewriting

7. Evaluation

To demonstrate the enhanced performance using our model, we will run a test scenario that has four pages in different 4 test cases, each page will output 50, 100, 150 and 200 Kbytes of data, and we chose this range of page sizes as most of normal web sites pages' size are within this range.

The first page is a static HTML page that contains the data inside it, same as case 1 in section IV, the second page is a dynamic PHP one that gets the data from a database server as case 4 in section IV, the third one will be the same as the last dynamic page but with "MySQL Query Cache" enabled, and the fourth page will be the same as the second but with our caching model applied. We will measure the average loading time of 100 consecutive requests for each case. Note that in the fourth page, the first request will load the dynamic file as a normal dynamic page, but at the end of the execution a cached version will be stored and each subsequent request will get the cached static page instead of loading the dynamic one once again.

To further illustrate the caching effect on the dynamic page, a comparison of loading time between a dynamic page before and after applying the caching using URL rewriting, note that at the first request the response time is high for the two pages, whereas at the second request the web server responded with the cached version of the page when applying our system, which dramatically decreased the loading time.

8. Conclusion

Web caching is an important subject since World Wide Web (WWW) had great popularity that grows every day. Many techniques have been developed to save network bandwidth and server processing load. We showed different types of web caching with the advantages and disadvantages of each type. We showed that caching at server side makes the cached data more controllable by the system administrator than caching at proxy servers or client-side browser. Then we proposed our model of a simple web caching mechanism based on URL Rewriting feature that is available in many web servers and showed by simulation how the model enhanced the response time of a dynamic web page.

REFERENCES

1. Wikipedia. (2009, June) Static Web Page. [Online]. Available: http://en.wikipedia.org/wiki/Static_web_page.
2. Wikipedia. (2009, June) Dynamic Web Page. [Online]. Available: http://en.wikipedia.org/wiki/Dynamic_web_page.
3. W3C, October 2009. HTTP/1.1, part 6: Caching, http://tools.ietf.org/html/draft-ietfhttpbis- p6-cache-08.
4. IIS, URL Rewriting Documentation. [Online]. Available:http://learn.iis.net/page.aspx/460/using-url-rewrite-module/. 2008.

5. Jaimie Sirovich, Cristian Darie, 2007. "Professional search engine optimization with PHP", Wiley Publishing, Inc., page 39-45.
6. Swaminathan Sivasubramanian, Guillame Pierre, Maarten van Steen and Gustavo Alonso, 2007. "Analysis of Caching and Replication Strategies for Web Applications". IEEE Internet Computing 11(1), pp. 60-66.
7. S. Sivasubramanian, G. Pierre, M. van Steen, and G. Alonso, 2006, "GlobeCBC: Content-blind result caching for dynamic web applications". Technical Report IR-CS-022, Vrije Universiteit, Amsterdam.
8. B. M. Subraya, 2006, "Integrated approach to web performance testing: a practitioner's guide", Idea Group Inc (IGI), page 32-41.
9. C. Bornhvd, M. Altinel, C. Mohan, H. Pirahesh, and B Reinwald, 2004. "Adaptive database caching with DBCache". Data Engineering, 27(2):11–18
10. Matthew Syme, Philip Goldie, 2003. "Optimizing Network Performance with Content Switching: Server, Firewall, and Cache Load Balancing", Pretice Hall, page 119.
11. K. Amiri, S. Park, R. Tewari, and S. Padmanabhan, 2003. "DBProxy: A dynamic data cache for web applications". In Proceedings of International Conference on Data Engineering, pp. 821–831.
12. Iyengar, Arun and Rosu, Daniela, 2002. "Architecting Web sites for high performance", Scientific Programming Journal., vol. 10, no. 1, pp. 75–89.
13. D. Davison, 2001. "A Web Caching Primer", IEEE internet computing, Volume 5, Number 4, pp. 38-45.
14. K. Rajamani and A. Cox, 2000, "A Simple and Effective Caching Scheme for Dynamic Content", Rice Univ. CS Technical Report TR 00-371.