Cache aware design for PHP-NUKE
以快取導向設計的 PHP-Nuke

Y. K. Chang and M. H. Hong
Dept. of CSIE,
National Cheng Kung University
{ykchang,p7691106}@mail.ncku.edu.tw

K. L. Chiang
Dept. of Information Management,
Chung Hua University
klchiang@mi.chu.edu.tw

摘要

近年來，隨著動態網頁的興起，許多棘手的問題也隨之衍生，如伺服器之處理能力被迫下降、網路之傳輸更加壅塞等等。雖然動態網頁的內容不一定會每分每秒改變，但是為了得到最新的資訊，快取機制會被關閉。

本研究提出讓動態網頁存成靜態檔案的方法來讓快取機制更有效率。我們使用了 Apache URL 重導方法與快取管理機制來增進我們的效能。我們把這套機制使用在 PHP-Nuke 中，然後比較原始的 PHP-Nuke 與使用我們方法的 PHP-Nuke 的表現。根據實驗的結果，修改過的 PHP-Nuke 的效能要比原始的 PHP-Nuke 好。

關鍵詞：動態網頁、快取、一致性、PHP-Nuke

Abstract

Over the past few years, dynamic web pages become popular[13]. This increases server load and makes cache ineffective. For dynamic generated pages, cache mechanism is disabled for obtaining newest data. But these pages may not change every time it is accessed or only a part of the page is changed.

In this paper, we propose an approach for writing cache-effective web applications to cache dynamic pages. Our proposed design improves the caching systems by using rewrite engine supported by apache web server and a novel cache management. We implement our concepts on PHP-Nuke, an popular news automated system, and compare the performance of original PHP-Nuke from our modified PHP-Nuke. Based on the results of the experiments, the modified PHP-Nuke performs better than the original PHP-Nuke.

Keywords： dynamic web pages, caching, consistency, PHP-Nuke.

1. Introduction

HTTP is used to transfer the Web documents. It was originally designed for browsing static documents. An important mechanism used for saving bandwidth, response time, and server load is to cache the documents. Using the original HTTP protocol, a static document that rarely changes can be attached with an “expiration time”. This tells the proxies and browsers that they do not need to reload the document from the server before expiration time. Additional validators, such as the content length, last modification time, entity tags, and cache control [7, 3, 18], are also needed in order to make them cacheable in the proxy servers.

However, during the last decade, the development of World Wild Web is gradually changing from static to dynamic. CGI、ASP、or PHP together with HTML forms allows dynamic creation of documents. These dynamic contents are constructed based on personalized service and request parameters at the time the document is requested. For those dynamically generated documents that may change on every request, cache mechanism is disabled—the expiration time is always set to “now”, sacrificing the benefits of caching.

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 in a period of time. So we can see 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 [9, 6, 2, 14].

In addition to the uncacheability of dynamic web pages, each request for dynamic page invokes a program, script or database access, which typically performs significant operators to generate the requested pages. It is the main reason why web server can be overloaded by only a small number of requests. If server’s service rate can’t keep up with increasing requests, the server's throughput will decrease and users will experience long response time or request timeout. So, there are some techniques proposed to tackle these drawback.

Delta encoding [12] observes that most dynamically generated documents have many fragments in common with their earlier versions. Instead of transferring the complete document, it updates cache entries by transferring only the differences, or ''delta,'' between the cached entry and the current new value. But the drawback of delta-encoding is that it requires protocols for management of past versions.

Active Cache [16] attach cache applet to each document which is a piece of code that can be executed by the proxy. The scheme needs proxies to invoke cache applets upon cache hits to execute the necessary processing without contacting the server. It saves network bandwidth but it also have drawback. It need to modify proxy server which is not practical to wide-spread use.

HPP [8] has extended HTML to allow the explicit separation of static and dynamic portions of a resource. It is also observes that dynamically constructed documents usually contain common constant fragments. The static portion contains instructions for inserting dynamic information. The static portion together with these instructions (the template) can be cached freely and dynamic part, binding file, are accessed every time.

In this paper, we will introduce PHP-Nuke 6.0 and improve its cacheability by using our mechanism. We observe that on PHP-Nuke only a small part of page is changed on every access. So The static portion of a page is separated from the dynamic portion where the static portions can then be cached, with (presumably small) dynamic portions obtained on each access. We propose a caching system to store the static part of dynamic generated web pages on disks and use rewrite engine supported by apache web server.

In section 2, we propose our cache mechanism. In section 3 we give brief description of PHP-Nuke. In section 4, we are going to introduce the way we use to modify the PHP-Nuke to be cacheable. In section 5, we show performance of original PHP-Nuke and modified PHP-Nuke with cache mechanism. In section 6, we show conclusions.

2. Proposed caching system

As we know the process of accessing a static file is much faster than that of accessing a dynamic web page. Therefore, our basic idea is

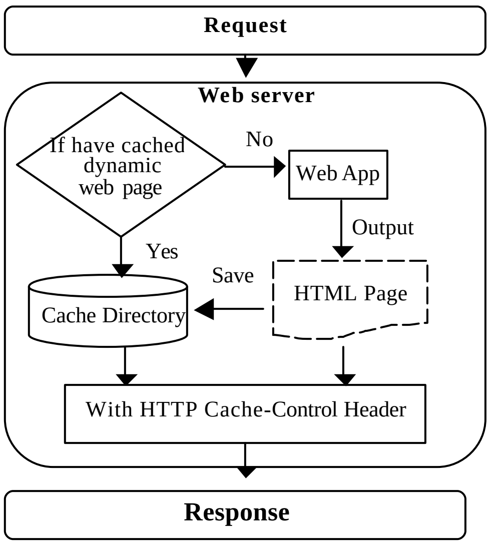

Figure 1.concept of process a request

to store the dynamic pages as static files in cache[10, 4]. The concept and dataflow is shown in Figure 1.

We put all cached files in a directory named Cache Directory. If the requested file is in the Cache Directory, we response the request with file in it. Otherwise, we first trigger WebApplication to generate requested files and attach with HTTP Cache-Control Header on reply. Then, we store the reply in Cache Directory as a static file. To implement our concept, components in the system are developed and shown in Figure 2.

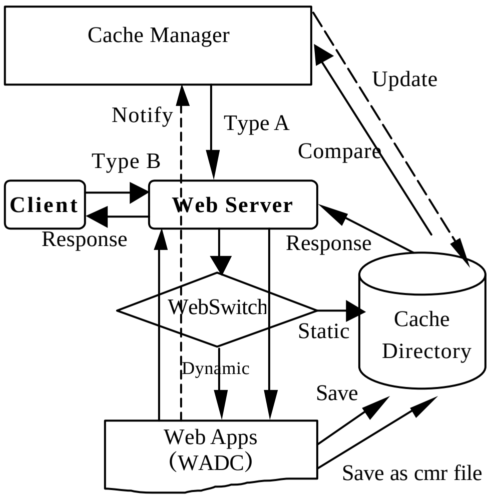

Figure 2. The structure of Dynamic Web page system

http://host/abs_path/page?k1=v1&k2=v2 (Type A URL)
http://host/abs_path/page!k1=v1&k2=v2.html (Type B URL)

Table 1：URL formats

2.1 Type B URL

In this paper, the format of URLs is

classified into two types as shown in Table 1. 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[19]. 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. 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.

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 3 shows the code of a sample program with two pairs of keywords and values.

```
<Script Language=JavaScript>   function Location() {
var UrlStr;
UrlStr = "http://host/abs_path,page! k1=" +
document.LocationBody.k1.value+"&k2=" +
document.LocationBody.k2.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 3: A javascript code to imitate the action of POST method.

2.2 Web Switch

We use URL Pre-processing to process request URL before Web Switch begins to work in Figure 4(put in the end of paper). In the beginning, URL Pre-processing checks if the request is for javascript file or not. If it is not to request javascript file, URL Pre-processing first checks if the requested URL exists. The requested URL could be for dynamic page or for static page or a cached file. So if the requested URL exists, URL Pre-processing passes it to Web Switch. If the request is for cached file, the format must be Type B. If file doesn't exist, URL Pre-processing converts URL into Type A, then passes it to Web Switch. Web Switch executes the normal process for dynamic page as follows: execute the corresponding program, generate dynamic page, and store the dynamic into cache file for later use. If file exists, Web Switch replies with the cached file in Cache Directory.

2.3 Web Application Design for Caching (WADC)

When the dynamic page is generated by dynamic page program, it will be dropped by the web server after it returns the response to the client. Also the dynamic page generated by dynamic page program will not contain any HTTP Cache-Control Header. So, it will not be cached by proxy. In this section we are going to propose how to make dynamic generate page cacheable.

2.3.1 Generate appropriate HTTP Cache-Control Header

WADC must generate appropriate HTTP Cache-Control headers according to changing frequency of dynamic page. These headers are attached to dynamic page before it is replied to the client.

2.3.2 Storing dynamic generated page into cache file

WADC must store the dynamic generate page as cache file in Cache Directory. The name for cache file is in Type B URL format. For example, if client request for /cachedir/calculate.php!v1=2&v2=3.html and this doesn't in Cache Directory. Then URL Pre-Processing will translate the URL into /calculate.php?v1=2&v2=3. Calculate.php first generates dynamic page, attaches it with appropriate Cache-Control headers, and stores the dynamic page into Cache Directory with the name calculate.php!v1=2&v2=3.html.

2.3.3 Informing Cache Manager

WADC must inform Cache Manager that the dynamic page is stored as cache file in Cache Directory and when to refresh that cache file. Cache Manager send Type A URL request to WADC to regenerate a cache file when the predefined time is expire. And regenerated cache file must save in TYPE B URL with .cmr in the end of the file name. Cache Manager then check if the original cache file and regenerated cache file are the same. If they are the same, delete the regenerated cache file, otherwise replace the original cache file with new generated cache file. The reason is if original and newly generated cache file are the same, we must keep the original cache file's last modified time unchanged to make it still cached by proxy when proxy send If-Modified-Since to check the consistency.

Cache Manager is in charge of the cache files in Cache Directory. It is behind the web server. Only WADC can inform it about how to manage the cache files. And Cache Manager will not response to WADC. The attributes of cache files maintained by the cache manager is as follows.

- ID: The name of cache file in the form of TYPE B URL.
- File size: in Byte.
- MD5 value[17]: used to determine the consistency.
- Last Modified Time: The last time the cache file is modified.
- Last Access Time: The last time the cache file is accessed.
- Expiration Time: The expiration time of cache file.
- Consistency Check Method: The method Cache Manager used to maintain cache files.

WADC tells Cache Manager that ID, Consistency Check Method and Interval Check Time of a cache file. Cache Manager will active get the File Size, MD5 value, Last Modified Time, Last Access Time and will use system time of web server and Interval Check time to calculate Expiration Time.

Database will change with time, so we will get different result on different time if we query with the same input arguments. Our Cache Manager offers three ways to maintain the consistency:

(1) Method for Regularly Changed Objects:

This method is designed for the web page that is changed in a fixed period of time. For example, the web version of the daily news paper is changed roughly every 24 hours. Hardware based sensor combined with a CGI script probing the field data in a fixed period of time is another example. Thus, the cached file will change regularly. So WADC will informs Cache Manager like this: "calculate.php!v1=2.html:T1800". This means that the cached file calculate.php!v1=2.html is regularly changed object and the change interval is 1800 seconds. Cache Manager will send Type A URL request to web server every 1800 seconds, store the newly generated file as "calculate.php!v1=2.html.cmr", check the MD5 of the new and original cache file. If they are the same, delete the new one, otherwise replace the original one with new one with the name "calculate.php!v1=2.html".

(2) Method for Irregularly Changed Objects with Notification:

This method is designed for the situation that the database manager and web master are the same person. Therefore, it is possible that the update process of the database can be designed in such as way that every time the database is changed, the cache manager will be informed. We use the method like Signal and Message Queue in UNIX system to let the database manager or program that will change data in database to notify Cache Manager to regenerate the corresponding cache file. For example, when WADC notify Cache Manager with "calculate.php!v1=2.html:N", Cache Manager with set the Consistency Check Method of that file as Method for Irregularly Changed Objects with Notification and when it receives "calculate.php!v1=2.html:N" again, it use the same procedure as mention before to send Type A URL request and check difference between original and new cache files.

(3) Method for Irregularly Changed Objects without Notification:

This method is designed for the situation that the database manager and web master are the different persons and it is not possible to force the database programmers to write the program in such a way that when the database is changed, the cache manager is informed. One reason is that the database manager may not know who is using the database.

The basic mechanism to obtain the change frequency is to adaptively probe the concerned data. Since this part is not related to the PHPNUKE in which all the web programs and databases are maintained by the same person, we will ignore the details in this paper.

Other than the consistency problem, the space constraint is also addressed in the design. If the free space is less than 10%, Cache Manager will start the swap procedure. The policy we use is LRU (Least-Recently-Used). When swapping, Cache Manger will retrieve the Last Access Time of all cache files, find the one with longest time without access and delete it until the free space is more than 10%.

2.5 Separate static part from dynamic part

We will modify PHP-Nuke and put the static part and dynamic part into javascript files[11]. When Web Switch receives a request like "xxx.js", if it exists, Web Switch replies with javascript file in Cache Directory. Otherwise, Web Switch executes the corresponding dynamic program to generate javascript file. We know which javascript file will change by writing corresponding code in PHP-Nuke to notify Cache Manager to refresh javascript file. Cache Manager can manage these javascript files by "Method for Irregularly Changed Objects with Notification" as we mentioned before.

```
RewriteEngine on
RewriteLog logs/rewrite.log
RewriteLogLevel 9
RewriteMap
urlparse    prg:/usr/local/apache2/cgi-bin/urlparse.pl
RewriteCond
/usr/local/apache2/htdocs%{REQUEST_FILENAME} ! -s
RewriteRule    ^/cache_dir(.*)\.html$      ${urlparse:$1}    [L]
```

Figure 6: URL rewrite Rule

```
#!/usr/bin/perl -w
$|=1;
while(<STDIN>) {
chomp $_;    s/,/\//g;    s/!/?/g;
printf $_."\n";      }
```

Figure 7 : urlparse.pl

2.6 Implementation

Web Switch:We use Apache mod_rewrite [1] to implement our URL Pre-processing. Apache mod_rewrite gets the requested URL before Apache services the request. Mod_rewrite uses Condition and Rule to rewrite URL and Apache will use this rewritten URL as client 's request URL and start to process it. Figure 6 and 7 are our implementations.

WADC: General speaking, HTTP Header must send to client before other content. We use ob_start() function in PHP to buffer the generated page and calculate page size. Then, attach HTTP Header with Cache Control Information and send HTTP Header to client before sending the page content.

3 .PHP-Nuke

3.1What is PHP-Nuke

We can say that PHP-Nuke is a Content Management System, Web Portal System, News automated system. The main objective of PHP-Nuke is to have an automated web site to distribute news and articles. It is designed for people to arrange their web sites in a flexible way. With PHP-Nuke, a web master can easily manage his web site through administrator pages. PHP-Nuke is totally written in PHP and requires Apache Web server, PHP and MySQL.

3.2 Features and Functions

The main features for users include: surveys, Topics, Web Links, Recommend web master, submit News, upload download links, and themes manager for registered users.

For administrator, PHP-Nuke has a friendly administration GUI with graphic topic manager. Graphic topic manager contains: option to edit or delete stories, option to delete comments, Refers page to know who link us, sections manager, and many
friendly functions.

3.3 Directory structure

```
html /───── admin /
     ├───── blocks /
     ├───── includes /
     ├───── language /
     ├───── modules /
     └───── themes /
```

3.4 Framework and structure

We introduce the structure of PHP-Nuke in two views: user view, and administrator view.

User View:

PHP-Nuke is composed of 3 column portals, the two lateral ones including the blocks, the central one displaying the functional modules. Beyond these 3 columns it also has a header (top of page) and a footer (bottom of page) that appear in all pages.

Blocks: these are present in the left/right columns of PHP-Nuke's portal and render functions that are repeated in all pages of the site (e.g. the Categories, search and login blocks).

Modules: They are the kernel part of the page, they are placed in the center column and each one has its own function. For example the news module renders the articles, the Feedback module let user to submit comments and suggestions to the administrator. We can see structure of whole web page in bold type and corresponding files in italic type with underline in Figure 8.

Header *Header.php*		
Header *Header.php* (Left Block)	**Modules** Modules.php?name=xx	**Header** *Header.php* (Right Block)
Footer *Footer.php*		

Figure 8

Besides blocks and modules, there is another function users can use is to change theme. Theme is used to present the format output. Users can change their theme which let them to have their personalized graphical interface.

PHP-Nuke separate the presentation from database query. For example, when a client request main page – /html/index.php, web application first get user's theme from cookie, query database to get the data, and then output the whole page by the user selected theme. So, if the user only change theme and the data is not change, we just need to reload the theme part.

Administrator View:

Once logged in, the administrator finds an interface that lists all the areas on which can be acted upon with GUI graphic topic manager. In this paper, we focus on how to modify User view, because administrator may contain few people and it won't bring about the server's load very much.

4 Modified PHP-Nuke

Dynamic generated pages may not change every time it is access or just a part of page is changed. So we are going to separate the static portion of a web page from the dynamic portion.

The static portions can then be cached for a longer time, with (presumably small) dynamic portions obtained on each access. In this section, we are going to present how to modify PHP-Nuke into static and dynamic part.

4.1 Basic concept

The language we use for combining static part with dynamic part in client side is javascript. That is to say, we convert the php code into javascript and save them into javascript files. Here we discuss the normal users, not the registered login user. We will discuss registered login user later. The concept we use is described as follows:

As we mentioned before, a whole web page contain (1)header(top page), (2)left/right blocks, (3)central modules, and(4) footer(bottom page). Header, left/right blocks, and footer appear repeatedly in every web page. Also, the output process contains two steps.

Firstly, when a user requests a page, PHP-Nuke determine what theme(default or login user selected theme in cookie) to use and includes /html/theme/*Selected_Theme* /theme.php.

Secondly, it includes corresponding php file to present central part and finally composes these two parts to output the whole page.

The following abstract structure is a php script that phpnuke uses.

```
<?php
$ThemeSel = get_theme(); //get the theme user choose
include("header.php");     //output header
include(" modules/ Selected_module /index.php");
//output the central part
include("footer.php");  //output the bottom of the page
?>
```

Example 1-1

The following abstract structure is the static html file we use.

```
<HTML> <HEAD> </HEAD> <BODY>
<script src='ThemeDefault_format.js'></script>
<script src= 'header_function.js'></script>
<script src= 'header_data.js'></script>
<script src= 'ModulesSelected_function.js'></script>
<script src='module_data.js'></script>
<script src= 'footer_function.js'></script>
<script src='footer_data.js'></script>
</BODY> </HTML>
```

Example1-2

We can see the corresponding javascript files in Figure 9.

We separate function part from data part because function changes less frequently than data part. For some xxx_data.js that may change frequently, we pick out the corresponding database query code in PHP-Nuke to write new PHP script. So, we just execute data generation and don't need to generate the whole page content like PHP-Nuke.

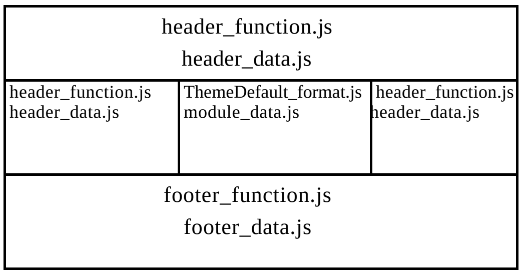

Figure 9

We translate the "judge part", "function part" in php language into our javascript form in xxx_function.js. And then put the query data from database into our xxx_data.js. For the ThemeDefault_format.js, it contains functions for header_function.js, ModulesSelected_function.js, and footer_function.js to call to format their theme forms. For header_data.js and footer_data.js, it is the same to all users except the online information in left block. We will discuss this later. For ThemeDefault_format.js, the descriptions in this example are for the anonymous non-login users. We will discuss registered login user later.

4.2 Advanced implementation

In this section we will discuss hyperlinks, registered login users and query strings that may change every time user request.

4.2.1 Registered login users:

The difference between registered login users and anonymous un-login user is that login user has cookie. We use Example1-2 to explain how to use the same html for different login users.

We use a javascript function to get user's cookie, and combine cookie with the filename in Example1-2 to form a new request filename.

GetCookie("name") is function we write in javascript to get cookie and we put it in **GetCookie_function.js**. We assume that the value get from GetCookie("name") is "*Cookie_value*", then the Example 1-2 is change to Example 2.

We add <scriptsrc='GetCookie_function.js'></script> to include our get cookie function. Different login user will get different ***Cookie_value*** and that looks like <script src='module_data***Cookie_value***.js'></script>.
The user theme may change with different user <script src='Theme**Select**_format.js'></script>.

We also add <scriptsrc='login_function.js '></script><scriptsrc='login_data***Cookie_value***.js'></script> to show login information in left column of page. After user logout we delete all the xxx_data***Cookie_value***.js for security reasons.

```
<HTML> <HEAD></HEAD> <BODY>
<script src= 'GetCookie_function.js'></script>
<script src= 'header_function.js'></script>
<script src= 'header_data.js'></script>
<script src= 'login_function.js'></script>
<script src= 'login_dataCookie_value.js'></script>
<script src='ThemeSelect_format.js'></script>
<script src= 'ModulesSelected_function.js'></script>
<script src='module_data Cookie_value.js'></script>
<script src= 'footer_function.js'></script>
<script src='footer_data.js'></script>
</BODY> </HTML>
```

Example 2

4.2.2 Hyperlinks and query string:

In PHP-Nuke the first layer of hyperlinks look like http://domain/modules.php?name=val.

So, its TYPE B form is http://domain/modules.php!name=value.html. First layer of hyper links presents the main categories of web site. For the second layer it may looks like http://domain/modules.php?name=value1&op=value2. So, its TYPE B form is http://domain/modules.php!name=value1&op=value2.html. Second layer means user has choose a category and start to read the corresponding articles.

4.2.3. Combine modified PHP-Nuke with Cache mechanism

For our modified PHP-Nuke we put our output in html file and javascript file. We use the same mechanism that we use in html file (in section 2) for javascript file. So we need to add rewrite rule for javascript files, see Figure 10.

Most part of PHP-Nuke web page can only be modified by Administrator or Survey that may change when a user submits a vote. For an administrator, he can modify Post News, Add Story, Change Configuration for web site, etc. For those javascript files that may change according to the Administrator's action or user submit a vote, they can classified into "Method for Irregularly Changed Objects with Notification" as mentioned before in section 2. That is to say, when the administrator updates database, change some part of web site or user submit a vote, we can write code in PHP-Nuke's corresponding files to notify Cache Manager to refresh the related javascript files.

```
RewriteMap
urlparse     prg:/usr/local/apache2/cgi-bin/urlparse.pl
RewriteCond
/usr/local/apache2/htdocs%{REQUEST_FILENAME} ! - s
RewriteRule   ^/cache_dir(.*)\.js$   ${urlparse:$1}   [L]
```

Figure 10: a sample apache rewrite rule.

5. Performance

A number of experiments are conducted to show the performance improvement of the proposed system. The **HTTPerf**[5] performance tool for web servers is used. We use PHP-Nuke 6.0 to test performance. We consider three pages. The first one is the homepage of our PHP-Nuke index.php and our modified PHP-Nuke with cache mechanism. The second one is the first layer links of PHP-Nuke, we use Feedback link as our experiment. The last one is the pages with query strings for articles in 2003 Aug.

The links are shown as follows.

A: http://domain/phpnuke/index.php and http://domain/phpnuke/index.html

B: http://domain/phpnuke/modules.php?name=Feedback and http://domain/phpnuke/modules,modules.php!name=feedback.html

C:http://domain/phpnuke/modules.php?name=Stories_Archive&sa=show_month&year=2003&month=08&month_l=_AUGUST and http://domain/phpnuke,modules.php!name=Stories_Archive&sa=show_month&year=2003&month=08&month_l=_AUGUST.html.

We first show the size of these three pages in Figure 11. As we can see that without cache (First Access), our modified page size is a little bigger than original page in A and B links. That is because we put some functions and decisions that PHP-Nuke use into javascript files. And if it contains more same form output, our page size is a smaller than original. The reason is that we use loop in function to output the same format data to prevent redundant transfer. But with cache (After First Access), only the small part need to be transfer .

Second, we show the server load on processing these three requests in Figure 12. We show the server load when the request rate are 5、10、15and 20 connections per second. For the three dynamic generated pages, the server load increase when the request rate increase. But for our modified PHP-Nuke, three html files, the server load is much smaller. The server loads of these three html files are almost the same. So they are located in the same place in our figure and can not be shown clearly.

Third we show the response time in Figure

13. We can see that for all six pages we test, the server loads do not increase when request rates change from 5 to 20, and the response times of our html pages are almost the same. But when the request rate is more than 30 per second the response time of the dynamic pages are greater than 1000 ms. That is because the server is overloaded, the response time will much longer than normal situation. For our modified PHP-Nuke, even the request rate is greater than 100 per second, the response time is smaller than 7 ms.

We also test the hit ratio of these three categories. For these three categories, they both use header.js (without login information) and footer.js. These two javascript files are rarely changed. So their hit ratio is almost 100%. But for login_data.js, it may change with user login or logout and different users use different login_dada.js, so hit ratio is about 30%. PHP-Nuke contains several themes. Theme**Select**_format.js will look like ThemeDeepBlue_format.js or Theme ExtraLite_format.js and these js files will not change until web administrator change them. So during our test, hit ratio of these Theme**Select**_format.js is 100%. For module_data ***Cookie_value.***js, the change rate may depend on what module we test. For example, when we test modules.php!name=Feedback.html, module_data ***Cookie_value.***js in it did not change, so hit ratio after first access is 100%. But for modules.php!name= Top 10 the hit ratio of module_data ***Cookie_value.***js is about 80 %.

We can see that performance of our modified PHP-Nuke is much better than the original PHP-Nuke.

6. Conclusions

In this paper, we proposed a cache mechanism for the dynamic web applications. The basic system is to cache the dynamically generated web pages as static files. The proposed cache system has a cache manager behind the web server for managing the cached static files. If a dynamic web page changes very frequently, we will try to separate it into a dynamic part and a static part. The static part can be stored in a static file and managed by the cache manager. Only dynamic part is generated every time a request is made.

The proposed idea is implemented in PHP-NUKE to show its superiority. By using the performance tools like HTTPerf, we show that the modified PHP-NUKE performs much better than the original PHP-NUKE in terms of response time and server load, and transferred file size.

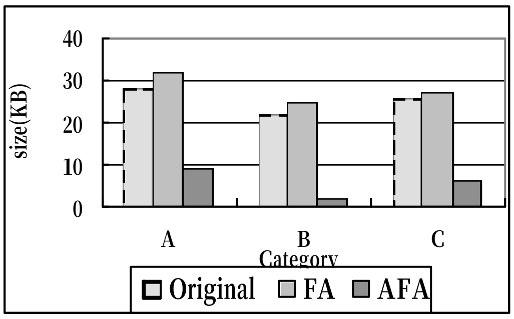

Figure11 .The size of original and modified PHP-Nuke

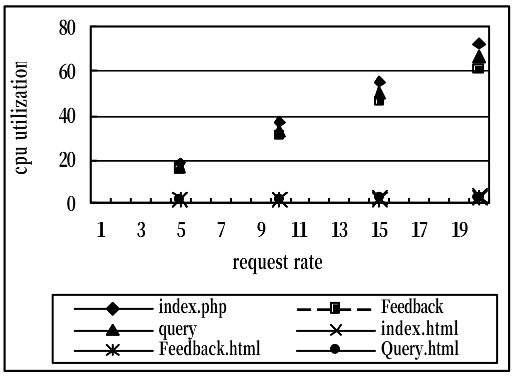

Figure12 .The CPU Utilization of original and modified PHP-Nuke

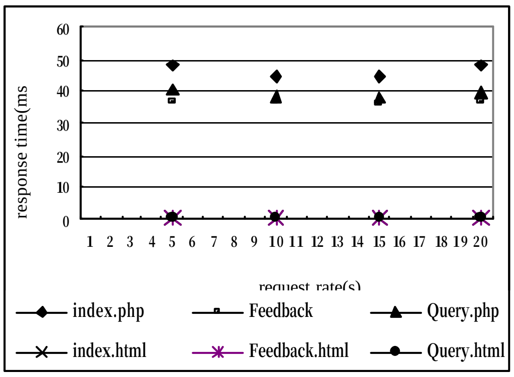

Figure13 .The Response time of original and modified PHP-Nuke

7. Reference

[1] Apache, "Apache HTTP Server Version 1.3: Module mod_rewrite URLRewrite Engine .

[2] 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

[3]Balachander Krishnamurthy, Jeffrey C. Mogul, David M. Kristol, "*Key Differences between HTTP/1.0 and HTTP/1.1*", *In Proc. of the WWW-8 Conference, Toronto*, May 1999.

[4] Ben Smith, Anurag Acharya, Tao Yang, and Huican Zhu, "*Exploiting Result Equivalence in Caching Dynamic Web Content*," in *Proceedings of Second USENIX Symposium*

on Internet Technologies and Systems(USITS99), Oct. 1999.

[5] David Mosberger, Tai Jin, "*httperf - A Tool for Measuring Web Server Performance*", *Workshop on Internet Server Performance, Madison, WI USA*, June 1998.

[6]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] Fielding,R.et.al., [HTTP1.1] "*Hypertext Transfer Protocol - HTTP/1.1*", HTTP Working Group,Internet-Draft, *draft-ietf-http-v11-spec-rev-03*, March 13, 1998.

[8] Fred Douglis, Antonio Haro, and Michael Rabinovich.*HPP:HTML macropreprocessing to support dynamic document caching*. In Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), December 1997.

[9] Gwertzman, J. and Seltzer, M., , "*The case for geographical pushing-caching*", Technical Report HU TR-34-94, Harvard University, DAS, Cambridge, MA, 1994.

[10] Huican Zhu, Tao Yang, "*Class-based Cache Management for Dynamic Web Content*", *IEEE Infocom*, 2001.

[11] Huican Zhu, Ben Smith and Tao Yang, "*Hierarchical Resource Management for Web Server Clusters with Dynamic Content*", *In Proc. of the International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS'99), pp. 198-199*, May 1999.

[12] 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.

[13] K. G. Coffman, A. M. Odlyzko, "*Internet growth: Is there a "Moore's Law" for data traffic?*", *AT&T Labs Research, http://www.research.att.com/areas/transport_evolution/internet.moore.pdf*, Jun 2001.

[14] 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.

[15]Official PHP-Nuke http://www.phpnuke.org/

[16] 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.

[17] R. Rivest, *The MD5 Message-Digest Algorithm*,http://www.ietf.org/rfc/rfc1321.txt, Apr. 1992.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "*Hypertext Transfer Protocol -- HTTP/1.1*",

[19]Squid Web Proxy Cache, http://www.squid-cache.org/

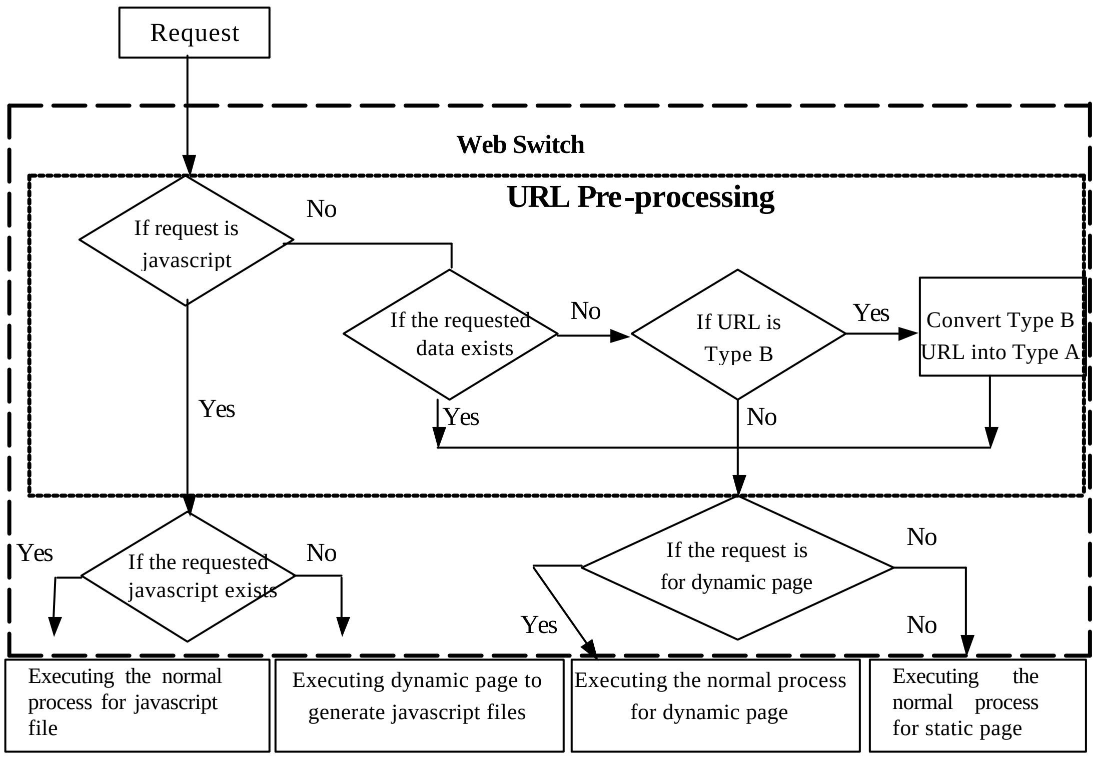
Request
Web Switch
URL Pre-processing
If request is javascript
No
If the requested data exists
No
If URL is Type B
Yes
Convert Type B URL into Type A
Yes
Yes
No
Yes
If the requested javascript exists
No
If the request is for dynamic page
No
Yes
No
Executing the normal process for javascript file
Executing dynamic page to generate javascript files
Executing the normal process for dynamic page
Executing the normal process for static page

Figure 4. Processing flow of URL Pre-processing and web switch