WUJNS
Wuhan University Journal of Natural Sciences

Vol.9 No.5 2004 828-834

Article ID: 1007-1202(2004)05-0828-07

Design and Implementation of A Dynamic Content Cache Module for Web Server

LIU Dan, GUO Cheng-cheng†, ZHANG Li

School of Electronic Information, Wuhan University, Wuhan 430072, Hubei, China

Abstract: Web offers a very convenient way to access remote information resources, an important measurement of evaluating Web services quality is how long it takes to search and get information. By caching the Web server's dynamic content, it can avoid repeated queries for database and reduce the access frequency of original resources, thus to improve the speed of server's response. This paper describes the concept, advantages, principles and concrete realization procedure of a dynamic content cache module for Web server.

Key words: dynamic content caching; network acceleration; apache module

CLC number: TP 393.09

Received date: 2004-05-20

Foundation item: Supported by the Science Committee of Wuhan

Biography: LIU Dan(1980-), male, Master candidate, research direction: high speed computer network, high performance server clusters system E-mail: laudan@eyou.com

† To whom correspondence should be addressed. E-mail: netcg@whu.edu.cn

0 Introduction

Nowadays, more and more attention is paid to the client's interactions with dynamic Web pages. In modern Web applications, such as E-commerce, online auction, search engine and etc. These applications must involve abundant database queries, thus induce the server's cost and network delay. Many studies have showed that the database query has become the main bottleneck of Web servers[1], so dynamic Web application is a severe limitation to Web site throughput. The server-side caching scheme allocates a block of memory in the server, then stores those hot query objects (mainly dynamic content) in the memory, consequently prevent repeated queries of the backend database. Cached objects are returned directly from the module after reconstruction when the same subsequent incoming client requests arrive at the Web server, which quickly improves the response time and the process capacity of the Web server.

Server-side dynamic content caching has many advantages:

1) Prevent repeated database queries and improve the response speed of the server.

2) Increase the throughput of the website and strengthen the scalability of the server.

3) Reduce the time used through I/O and the system's cost.

4) Easy implementation and management.

5) Excellent means of exploiting the server's potential performance, which can reduce the expenditure of hardware updates.

In the past, the research about caching is mainly focusing on the client-side caching and the proxy caching. With the

rapid development of Internet, dynamic Web application and the need for server-side high process ability, the focus is little by little shifting to the server-side caching. The study of Web server logs shows that user's access has a significant amount of temporal locality: that is, a significant percentage of the queries have been submitted more than once by the same or a different user. Although Web services often get updated, compared to non-caching approach, it would be possible to get great performance improvement by caching dynamic content, because recent studies have shown that much of the dynamic content is characterized as pseudo-dynamic[2], i. e., a dynamic composition of stored or static data. Now this field is limited within the discussion of the theory and lacks the specific implementation model and framework. The module in the paper is featured with easy implementation, complete functions, low overhead and high performance. The paper can provide some reference through the introduction of the server-side caching module design.

1 Design and Implementation of The Dynamic Content Cache Module

1.1 Design Goals of the Dynamic Content Cache Module

The server-side dynamic content caching is implemented with the module of Apache, a widely applied Web server. There are two main design goals: ① It can cache dynamic webpage objects and store them in memory, and use them to re-construct HTTP responses to answer incoming client requests. ② Basic caches management, including how to replace cached content and how to keep cached responses consistent with the content residing on the Web server.

Whenever any request for dynamic content arrives at Apache, it will get transparently handled by the cache module. The following will introduce the design and implementation of above functions[3].

1.2 Simple Introduction to Implementation Platform

Fig. 1 presents the Apache's architectures. Apache mainly consists of two parts: ① Apache core : Apache core defines the rules and steps(Apache break the request handling task into several phases) of how to handle requests and executes basic processing for each phase of request handling. ② Apache modules: a module is an active agent that can interact with Apache master or child servers to perform an action on certain events, and it can

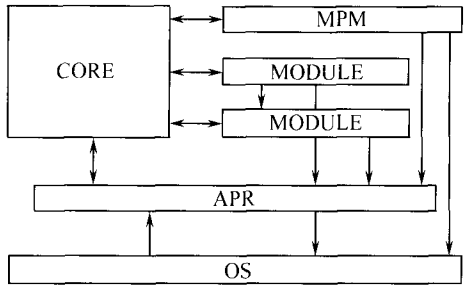


Fig. 1 Apache Architecture

customize how Apache functions at the certain phase (here we customize content deliver phase). The module is mostly composed with some init functions and handlers. The handler is also a function that is registered for some phase to add or overload Web server functionality. The module can have several handlers that are the main routines to cope with the real work. Our server-side dynamic content cache is implemented as Apache module[4,5]. To add certain functionality, just need to load the module into Apache server directly.

1.3 Architecture Design

The theory of the server-side cache module, to be brief, is to cache the responses of previous requests and use them to reply subsequent requests for the same content. It classifies requests into two categories[6]: cacheable requests and uncacheable requests.

The work going on in the following figure is all about the cacheable requests, yet the uncacheable request is straightly declined by the module immediately after being judged. Fig. 2 illustrates the module's architecture design.

1.4 Judgment Unit Design

This Unit has to make a judgment for two objects: one is the incoming HTTP request, to judge whether to rebuild a response from the caches directly; another is the response from the Apache Web server, to judge if it can be cached. HTTP/1. 1 offers several ways to mark an object uncacheable or cacheable. Detailed technical rules can be referenced by RFC 2616.

1.5 Storage Unit Design

The caches use in-memory hash table as its storage data structure, and the number of hash buckets can be customized. It works like this: firstly calculate the key value in the hash table via a hash function according to the URL of the request, if there is no conflict, map the caching object to a specific hash bucket. When conflict happens, adopts the chain address rule to deal with this

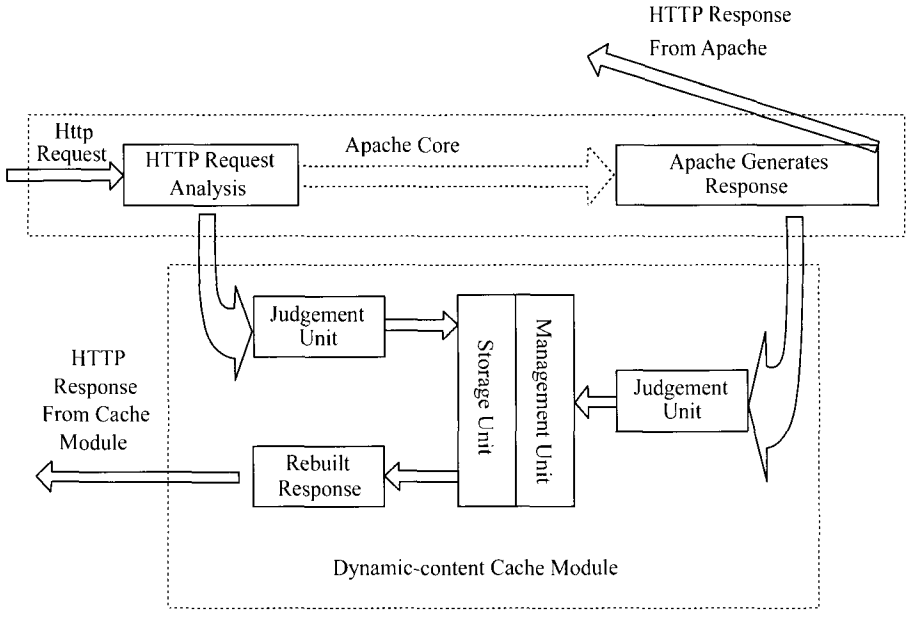


Fig. 2 Dynamic content cache module's architecture design

issue (firstly search along the conflict chain to find whether there is the object having the same name. If exists, update the hash bucket; if none exists, insert a new object at the end of the conflict chain). Such data structure scheme improves the efficiency of update and search.

In Linux, Apache has two work modes, prefork (multi-processing mode) and worker (hybrid multi-processing and multi-threaded mode)[4]. If Apache is configured with the prefork mode, taking the multi-process nature into account, we need a block of cache memory that should be shareable between processes to store objects; if Apache is configured with multi-thread worker mode, the threads themselves are featured with sharable memory, and so the issue of making memory shareable between processes can be out of consideration.

To simplify the implementation, we configure Apache with the worker mode, which is the premise of the implementation of the module.

1.6 Management Unit Design

1.6.1 Algorithm of consistency and replacement

Consistency is a complicated problem. Different websites, different content and applications need different algorithm of consistency. As a matter of fact, complete consistency is impossibly implemented[7]. In the paper, when a response generated for a dynamic content request passes through the cache module, meanwhile the module needs to set an expiry date for the request. In the event that a document does not provide an expiry date but does provide a last-modified date, an expiry date can be calculated based on the time since the document was last modified. The cache module configuration file can specify a factor to be used in the generation of this expiry date according to the following formula:

expiry-period = time-since-last-modified-date * factor

expiry-date = current-date + expiry-period

For example, if the document was last modified 10 h ago, and factor is 0.1 then the expiry-period will be set to $10 \times 0.1 = 1$ h. If the current time was 3:00 pm then the computed expiry-date would be 3:00 pm + 1 h = 4:00 pm. If the expiry-period would be longer than that set by CacheMaxExpire in the configuration file, then the latter takes precedence. When a cacheable request arrives after the expiration time, the cache module will hand the request to Apache for response, instead of returning the cached object.

Replacement is also a troubling issue; its implementation is website-specific dependent. The cache module implements two policies of replacement, LRU and Greedy Dual-Size, the latter is the default configuration. Greedy Dual-Size calculates a priority for each cached object according to its size and access frequency. When the cache memory's usage exceed the configured volume, the module replaces the content based on each object's priority value to load new objects.

1.6.2 Configuration management unit design

In order to achieve easy management and configuration of the module, the module defines many configura-

tion options, such as the default expiration time, last modified factor, max expiration time, max object count, max object size, min object size, replacement algorithm, cache size etc. Just need to write these options into Apache's configuration file and then restart the Apache server, the options will take effect immediately.

2 Program Implementation

2.1 Logic Flow of the Program

The logic flow of implementation is illustrated in Fig. 3, the following section gives a detailed description of the work mechanism of the cache module.

The cache module analyzes the requests from the Apache core.

If it is a GET (request method) request and is permitted to return response from the caches (if not, the module passes the request to another module registered with the same phase to deal with), the module takes the URL as the hash keyword and checks for a local copy.

If there is not a matched item, the response will be fetched from Apache. Through the route of the response returned to the client, the response content will pass through the cachein filter of the module. At this point, if the response is judged that it can be cached, the cachein filter will keep a copy (including the headers information of request/response, and the response entity) and update the priority queue of cached objects at the same time. Thus, when subsequent requests of the same URL arrive, the module can use the copy to reconstruct response for the client.

If the matched item exists, the module needs to fetch the header information of the copy to judge whether the copy is fresh, meanwhile see if the copy satisfies the client's requirements according to the header of the request.

If the copy is fresh and accords with the client's demand, the module will assemble the response with the cached copy plus the header information and then return it to the client through the cacheout filter;

But if the copy expires, the module don't simply pass the client request to Apache Web server, it still has to verify that if the request is a conditional one(request that contain If-modified-Since or If-Match or If-None-Match and etc HTTP headers) Validation may be performed by means of a conditional GET message to the or-

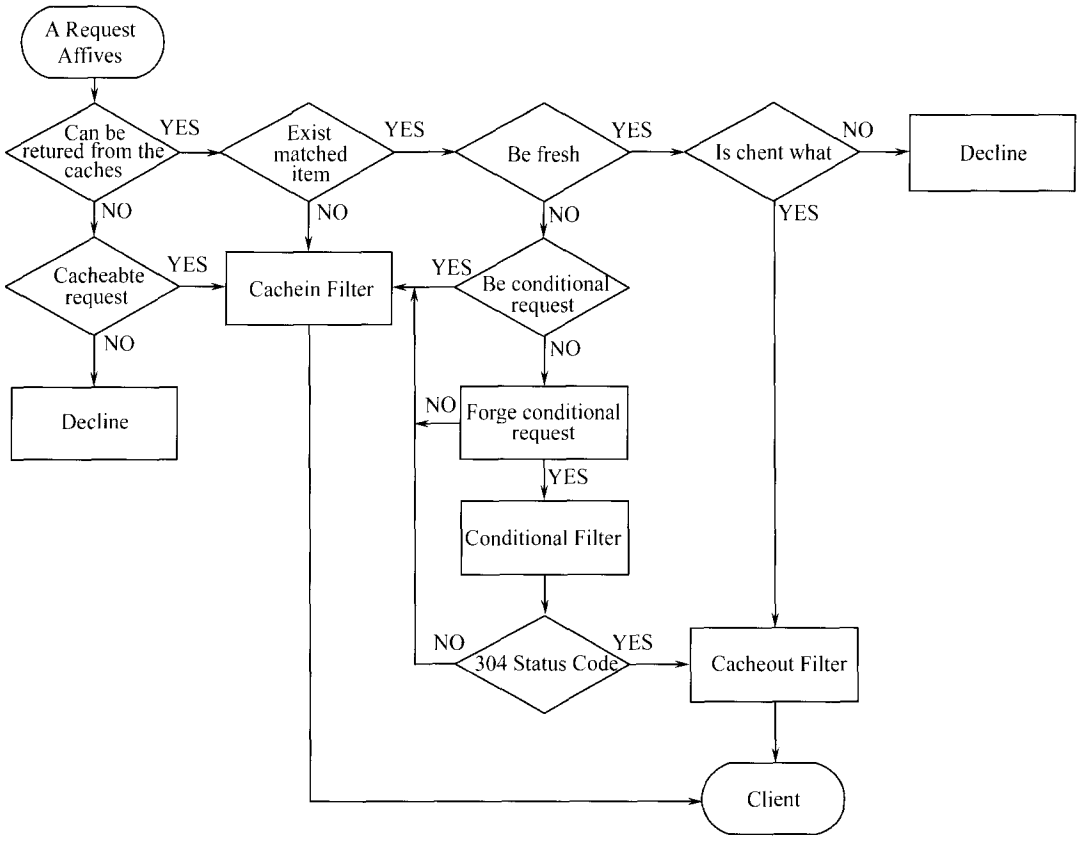


Fig. 3 Logic flow of program implementation

igin server, including a validator, such as the Last modified time of the resource, that allows the origin server to check whether the entity has changed[8,9].

If it is conditional, put the cachein filter into output filter chain, and then according to the returned status code from the Apache server determine to update the copy entirely or just keep the copy entity but updating the statistics of the copy.

If it is not conditional, forge a conditional request and put the conditional filter into output filter chain, then submit it to Apache server to revalidate the copy. If the returned status code from Apache is 304(not modified), pass the copy entity to the cacheout filter for assembling response to the client; if the copy is modified, don't return the response to the client until the cachein filter having completely updated the copy.

In the above three output filters, the conditional filter is only a judgment unit, all the real work is done by the cachein filter and cacheout filter. The cacheout filter's work is also straightforward for it just assemblies response using the copy and its statistics information. Yet compared with the former two filters, the cachein filter's work mechanism is the most complicated. Because it must strictly obey caching rules and has the duty to maintain and update the statistics information of cached copies. The next section gives the work flow figure of the cachein filter.

2.2 Work Flow Figure of the Cachein Filter

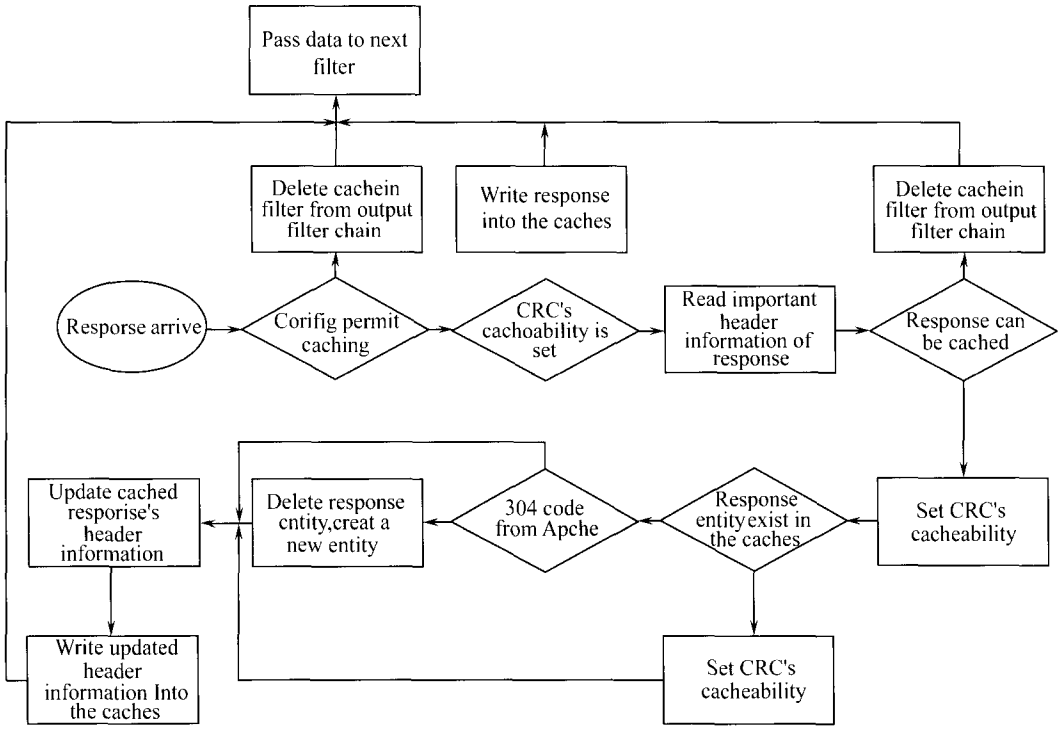
The work flow of the cachein filter is shown in Fig. 4, and the explanation of work mechanism is omitted here.

2.3 Key Data Structures

There are two key data structures in this module: cache_server_conf and cache_request_conf, which are used for maintaining the control and statistics information of the caches.

1) Cache_server_conf is the global configuration data structure of the whole module, is used for maintaining the control information of the caches, such as whether allowed to cache, the types of URLs allowed or forbidden, the default preserve time of the cached copy, the expiration factor of the copy etc.

2) Cache_request_conf is a data structure for the client request. It records all kinds of information about the resource requested by the client, including whether the copy is fresh, the hash address that the copy stores in, expiration time, last-modified time, ETAG identifier, CONTENT_TYPE, the length, last request time and last response time and etc, these options are used to maintain the statistics information of each cached copy.



CRC-Cacho_request_cont:a data struchuro for the client request

Fig. 4 Logic flow of Cachein Filter Program Implementation

Utilize these information to construct the header of the response and judge whether directly return the cached copy to the client.

3 Experiments and Results

1) Experiment environment: The server machine has an Intel 800 Mhz processor, 128 M SDRAM; the client machine has an Intel 800 MHZ processor and 128 M SDRAM. They are connected through 100Mbps Ethernet LAN. Redhat 9.0 is the operation system.

2) Application software: We use Apache 2.0.47 as our Web server configured with the worker MPM module to share cache memory. PHP 4.3.4 is used to provide server-side scripts for generating dynamic content. MySQL 4.0.16 is our database server.

3) Test software: We use httperf[10] as the test software, the benchmark can measure many performance values of the server.

Among the above test results, the most important measurement is "response time".

In Table 1 you can see that the value of this measurement is respectively 14.1 ms and 3.2 ms, 13.6 and 2.9 ms, 15.0 and 3.7 ms; and the performance has been improved by 77.3%, 78.68% and 75.33% separately.

And Table 2 when circularly request 50 different pages, the value of this measurement is respectively 16.2 and 3.1 ms, 13.1 and 3.3 ms, 14.8 and 4.6 ms, and the performance has been improved by 80.86%, 74.81% and 68.92% separately.

Table 1 Test results of requesting the same page repeatedly by varying ways (the page size is about 34 K)

Test mode	Module loaded	Persistent time/s	Connections/s	Requests/s	Average responses/s	Response time/ms	Transfer time/ms
1000 : 1	No	53.937	18.5	18.5	18.4	14.1	38.7
mode 1	YES	37.44	26.7	26.7	26.4	3.2	31.7
3000 : 1	NO	159.299	18.8	18.8	18.8	13.6	38.6
mode 2	YES	105.779	28.4	28.4	28.3	2.9	31.0
500 : 10	NO	286.456	1.7	17.5	17.4	15.0×10^3	42.2
mode 3	YES	169.918	2.9	29.4	29.5	3.7	30.2

In "test mode" column, the number ahead " : " shows the connections that httperf initiates; and the number behind " : "shows the requests sent through each connection. All the requests are sequentially sent(namely after the former request is finished, just continues the neon request)

Table 2 Test results of requesting different 50 pages circularly by varying ways (the page size is about 34 K)

Test mode	Module loaded	Persistent time/s	Connections/s	Requests/s	Average responses/s	Response time/ms	Transfer time/ms
1000 : 1	No	53.329	18.1	18.1	18.1	16.2	38.1
mode 1	YES	38.150	26.2	26.2	26.1	3.1	33.6
3000 : 1	NO	155.326	19.3	19.3	19.3	13.1	37.5
mode 2	YES	112.934	26.6	26.6	26.6	3.3	33.0
500 : 10	NO	294.447	1.7	17.0	17.0	14.8×10^3	44.0
mode 3	YES	198.829	2.5	25.1	25.3	4.6	35.0

In "test mode" column, the number ahead " : " shows the connections that httperf initiates; and the number behind " : "shows the requests sent through each connection. All the requests are sequentially sent(namely after the former request is finished, just continues the neon request)

4 Conclusion

This paper addresses the entire implementation of the server-side dynamic content cache module and its design details. Through the test it is showing that because the module tries its best to avoid repeated database queries, the server's performance increases a lot, especially the server's response time differs from no cache module nearly by a order of magnitude. This result approaches what we expected, implements the dynamic content cache, strengthens the scalability of the Web server. The next work we should do is to make further exploration and improvement to algorithm of consistency and replacement, optimize the structure of the procedure, thus to advance the consistency and the hit rates of the cache content as well as to excavate the performance of the server more deeply.

References

[1] Shawn. Does Your Server Needs Updating?. *http://www.yesky.com/ServerIndex/77132952596643840/20030305/1655286.shtml*, 2003-03-09(Ch).

[2] Arun I, Jim C. Improving Web Server Perfomance by Caching Dynamic Data. *USENIX Symposium on Internet Technologies and Systems*, Monterey, California, 1997. 49-60.

[3] Cory K J. Chi M. *Apache HTTPD Architecture and Evolution*, *http://apache hpi uni-potsdam. de/*, 2002.

[4] The Apache Software Foundation. *Developer Documentation for Apache 2. 0*, *http://httpd. apache. org/dos-2. 0/developer*, 2003.

[5] Kevin O. *Apache Module Development Tutorials*, *http://threebit. net/tutorials/*. 2003-07-01.

[6] Karthick R. A Simple and Effective Caching Scheme for Dynamic Content. *Rice University Computer Science Technical Report*. TR 00-371, Houston, Texas 2002.

[7] Jim C, Arun I, Paul D. A Scalable System for Consistently Caching Dynamic Web Data. *Proceedings of the 18th Annual Joint Conference of the IEEE Computer ahd Communications Societies*. IEEE Press, 1999. **1**: 294-303.

[8] David G, Brian T, Marjorie S, *et al*. *HTTP: The Definitive Guide*, Sebastopol, California: OReilly. 2002; **7. 8**: 150-200.

[9] Fielding R, Gettys J, Mogul J, *et al*. *RFC2616: Hypertext Transfer Protocol——HTTP/1. 1. http://www. w3. org/Protocols/HTTP/1. 1/rfc2616. pdf*, 1999.

[10] David M, Tai J. Httperf: A Tool for Measurin Web Server Performance. *Performance Evaluation Review*, 1998, **26**: 31-37.

□