高品質インターネット　4－5
(2002.　7.　26)

# 応答遅延時間に基づいたプロキシ・キャッシングの実験的研究

QusaiAbuein[†]　渋沢　進[†]

† 茨城大学工学部　〒316-8511 茨城県日立市中成沢町4-12-1
E-mail: †{abucinq,sibusawa}@cis.ibaraki.ac.jp

あらまし　インターネットデータへのエンドユーザからの要求は，しばしば大きな遅延や変動しやすい遅延をもたらすが，ユーザは遅い応答を好まない。この論文では，クライアントが要求したデータを受け取るまでに待つ時間を減らすアルゴリズムについて考察している。このアルゴリズムでは要求を返すのに必要な時間に応じてオブジェクトをキャッシュしており，ダウンロード時間の大きなオブジェクトをキャッシュし，時間のかからないオブジェクトをキャッシュしない。これによって，クライアントの待ち時間が減少できることを実験している。また，クライアントが要求したデータを受け取るのに必要な待ち時間の合計を減らしながら，高いヒット率とバイトヒット率を維持するアルゴリズムを導入している。ログファイルを用いて多くのアルゴリズムについて実験している。

キーワード　プロキシ・キャッシング，置換アルゴリズム，応答遅延時間，ヒット率

# An Experimental Proxy-Caching Study Based on Response Delay Times

Qusai ABUEIN[†] and Shibusawa SUSUMU[†]

† Faculty of Engineering, Department of Computer and Information Sciences, Hitachi, Nakanarusawa 4-12-1, Ibaraki, 316-8511 Japan
E-mail: †{abucinq,sibusawa}@cis.ibaraki.ac.jp

**Abstract** Each Internet data request from the end-user can lead to large and variable delays, such slow response times make Internet use undesirable and impractical for the end-users. In this paper we consider an algorithm to reduce the time that clients wait to receive their requested data. Caching by this algorithm is in accord with the time needed for a request to be answered, that is, objects that take more time to download are cached while those objects where downloading takes less time are replaced. We show that caching according to the length of object-download time reduces waiting time at clients. We also introduce an algorithm that achieves a high hit ratio and byte ratio as well as reducing the amounts of time clients have to wait to receive their requested data. Experiments for several algorithms were carried out with a variety of log files.

**Key words** proxy caching, replacement algorithm, response-delay time, hit ratio

## 1. Introduction

The explosive growth of the World Wide Web in recent years has led to huge increases in the amounts of network traffic that flows in response to HTTP requests, such that schemes to reduce the amount of network traffic have become necessary. Web caching is a popular way of achieving such reductions, and it is widely applied in web servers. Requests for data from popular servers frequently exceed the server's capacity to supply data; this leads to poor network latencies. Proxy servers are applied as intermediaries between browser processes and web servers on the Internet to mitigate this problem [13].

A proxy server provides documents from its local disk cache, eliminating the need to connect with the remote hosts that hold the documents, using the proxy server as a cache for web objects that would otherwise have long network latencies is an effective approach [1], [6]. Once an object has been transferred to the proxy, subsequent requests are handled by the proxy, so this approach achieves the following advantages :

- Reduced network traffic
- Reduced loads on Internet servers
- Reduced latency, clients notice the improved response times
- Reduced network costs [4].

Since the amount of storage space on a proxy is limited and clients continually request data, the cache of the proxy becomes full from time to time. The proxy's effectiveness is then maintained by removing objects from the cache to make space for new coming ones. An algorithm for control of this operation is known as a replacement algorithm or sometimes as a removal algorithm [2], [7].

The operation of most of the algorithms that are introduced to date has been based on measurements of the hit rates and byte-hit rates (the ratio of the number of bytes for objects that were hit to the total number of requested bytes), since their main aim is to reduce loads on the server and amount of network traffic, more information [5]. A few algorithms that are intended to reduce the length of time that clients have to wait to receive requested data have been proposed [9].

Users do not like to wait for web pages to load into browsers, and a few seconds may sometimes be too long wait for the client. Algorithms that take the length of the time taken to download data from their original locations into account in the saving of the objects in the cache are applicable to reducing the time that clients spend waiting to have their requested files. Saving objects according to the length of fetching time alone only leads to low waiting time for clients, but not to high hit ratios and byte-hit ratios. A further algorithm in which the multiples of the number of requests and the length of download time for objects is considered leads to an increase in the hit ratio and byte-hit ratio while still lowering the waiting time for clients. In this paper we introduce an algorithm that is intended to reduce the length of time that clients have to wait until they receive all of the data they have requested. For each requested object, we measure the time needed to fetch that object from its location, then cache those objects for which the times are long to reduce the client-waiting times. We also measure the hit ratio and byte-hit ratio. We then perform several experiments with another algorithm which considers the multiple of the number of requests and the download time for each object. The results of the new algorithms are compared to those of such other algorithms as LRU, FIFO, LFU, and SIZE.

The organization of this paper is as follows: the next section is a description of our preliminary studies; this is followed by a description of two response-delay-time algorithms. The next three sections report on our experimental environment, experiments and results, respectively. This is followed by discussion and analysis our results, and we give a brief summary of this work and our future intentions in the conclusion.

## 2. Preliminaries

Whether the goal of the caching and replacement algorithm is to increase the hit ratio or byte ratio or to reduce the response time, the size of the requested message plays the main role in achieving the goal, since large messages take more space in the memory and longer time to be downloaded.

### 2.1 Service Time and Message Size

The response time is measured as the time the message or request leaves the client until the response comes back, it passes through several networks and routers. Response time depends on the message size, the overhead and the bandwidth of the networks [12]. We can notice that the message size is variable and it differs from request to request.

As a consequence, the service time $T_{service}$, which is the time between a message leaving the client and the arrival of the corresponding response, is directly proportional to message size $G$:

$$T_{service} = bG + d \qquad (1)$$

Where $b$ and $d$ are constants.

With caching, the equivalent measure is the total response delay time $T$, which is the total time the clients have to wait before receiving their requested data, and the message is referred to as an object; $T$ is also direct proportional to the size $S$ of the object:

$$T = aS + c \qquad (2)$$

Where $a$ and $c$ are constants. Therefore, the message size is one variable that affects the total response-delay time. We will show by experiments that there is a relation between the response-delay time, the number of messages, and the hit ratio.

### 2.2 Hit ratio and Byte-Hit Ratio

The efficiency of a caching algorithm is often measured by the hit ratio or byte-hit ratio (byte ratio) of the cache. The hit ratio is the number of cache hits divided by the total number of requests for objects. The following equation gives the hit ratio:

$$hit\ ratio\ = \frac{number\ of\ requests\ served\ from\ the\ cache}{total\ number\ of\ requests} \qquad (3)$$

The byte ratio is the number of bytes transfered over the number of byte requested, that is, it represents how much bandwidth the cache is saving. The usual way to compute the byte ratio is as follows:

$$byte\ ratio = \frac{total\ size\ of\ objects\ serviced\ via\ cache\ hit}{total\ size\ of\ requested\ objects} \qquad (4)$$

## 3. Response-Delay-Time Algorithms

### 3.1 The Basic Algorithm

The time a client has to wait to receive its requested data is called the *response-delay time*. With a response-delay-based caching algorithm, objects are saved in the cache according to the time taken to fetch the objects from their original locations to the client browsers. The object with the longest such time is saved at the top of the cache, while that with the shortest time is saved at the bottom of the cache; replacement occurs from the bottom of the cache. We call this caching algorithm the *basic response-delay-time-based algorithm*.

## 3.2 Accumulative Algorithm

We may also consider the number of object requests in addition to the response-delay time of the object in either of two ways. One way is to consider the number of requests as a second criterion, while the other is to multiply the number of requests by the length of response-delay time and then use the resulting value to judge whether or not the objects should be saved. The reason for this is explained in the following discussion:

The efficiency of the algorithm is measured by the total time that clients have to wait to receive responses. The total response-delay time is computed for each request and applied so that those objects with large response-delay times are cached, while those with short response-delay times are replaced, to be fetched again when they are requested. Taking the number of requests for each object into consideration increases the efficiency of the algorithm. Consider the following example:

**Example:** Table 1 shows requests over a period of time:

表 1 Some requests over a period of time(all values are time in sec)

| Object No. | Response-Delay Time ($t$) | No. of Requests ($n$) | Accumulative Response-Delay ($t * n$) |
|---|---|---|---|
| $O_1$ | 20 | 5 | 100 |
| $O_2$ | 10 | 7 | 70 |
| $O_3$ | 8 | 9 | 72 |
| $O_4$ | 7 | 20 | 140 |
| $O_5$ | 4 | 5 | 20 |
| $O_6$ | 2 | 35 | 70 |

An object will be placed in the cache until it is full; suppose the entry of $O_5$ fills the cache. At that time, the objects will have been placed in the cache in order of response-delay time, that is, $O_1, O_2, O_3, O_4$, and $O_5$, in that order, are in the cache.

When $O_6$ is requested it will not be placed in the cache because its response-delay time is shorter; it will thus be fetched every time it is requested. Table 1 also contains the accumulative response-delay times, which are the multiples of the number of requests and the response-delay times for the respective objects. Note that, with the accumulative response-delay ($t * n$) as the criterion, $O_5$ will be replaced by $O_6$ and the total response-delay time will be reduced by the difference between the total time needed to fetch $O_6$ and the total time needed to fetch $O_5$, that is, $70 - 20 = 50$ seconds.

Taking the number of requests for each object into account thus keeps the total response-delay time reduced, that is, it increases the efficiency of the algorithm; this approach also leads to increased hit and byte ratios, because objects that are more frequently requested are more likely to be kept in the cache.

This scheme assigns a field that contains the multiple of the number of requests and the response-delay time for each object in the cache. This field is updated each time the object is requested, and the sorting of objects in the cache is according to the values in this field. The object with the largest accumulative value will be at the top of the cache while that with the smallest value will be at the bottom, and will be replaced first. We call this caching algorithm the *accumulative algorithm*.

**Algorithm**(Accumulative algorithm)

While there is a request, the algorithm checks whether the requested object is placed in the cache. If it is placed at the cache then it increases the number of requests, conducts the multiple of the number of requests by the fetching time. Then add the result to the total response time, and it resorts the objects in the cache according to the descending order of the calues. But if the requested object is not in the cache then it adds the fetching time to the total response time and checks whether to cache the object. If it is to be cached then it checks for enough space, otherwise it replaces the object, and (last) resort the cache.

**Input:** objects $O_i$ from clients.

**Output:** object fetching time $t_i$, the number of requests $n_i$, the multiple $p_i$ of $t_i$ by $n_i$, total response time $T$. The cache contains objects sorted in the descending order of the multiple $p_i$.

```
0. $T = 0$;
While ( there is $O_i$ ){
1. if $O_i$ is in the cache then
        $n_i++$,
        $p_i = n_i * t_i$,
        $T+ = p_i$,
        resort according to $p_i$;
2. if $O_i$ is not in the cache
        $T+ = t_i$,
        if $n_i * t_i > n * t$ of last object in the cache then
          if space is enough then
            cache $O_i$
          else
            replace $O_i$;
        resort according to $p_i$;
}
end.
```

## 4. The Experimental Environment

The log files of today's proxy servers, such as the *Squid* and *Netscape* systems, indicate the time spent on each request [3]. This is, of course, useful in terms of finding out how long clients are waiting to receive the data they have requested. Our study and experiments carried out on a Pentium 3 PC, running version 4.4 of the freeBSD operating system, using a *Squid* [8] proxy server to generate log files, and algorithms implemented in the C language. We needed to generate a log file instead of using an existing one, such as the Berkeley log file(注1), or the log file of the DEC research center(注2), is that those log files, for security reasons, do not include the complete requested URLs. We need the complete URLs to generate lists that include the response-delay time for each request in a specific period of time for reference when requests are received.

We chose *Squid* software because it is free software, works under the Unix operating system, is high-speed, and logs the time needed to fetch each request. Squid acts as an agent, accepting requests from clients such as browsers and passing them to appropriate Internet servers [11].

We thus downloaded *squid* 2.4 [10], installed and configured it on our PC, and connected the computers of our laboratory to the resulting Squid-based server. We used a program called *client*, the execution of which with *Squid* software is supported, to fetch requests for access reissue them, thus generating requests for repeated access.

The creation and use of the log file were according to the following steps.

Step 1: Installing the Squid software on the PC and connecting an other computer, which runs the browser iCab, to the PC. This is shown in Figure 1.

Step 2: Extracting a file that contains unique URLs and the download times for the information at those locations from the log files generated by squid, as is shown in Figure 2.

Step 3: The replacement algorithms read the requests from the generated log file and read the corresponding download times from the extracted file when the object is not in the cache, as is shown in Figure 3.

## 5. Experiments and Results

The log file of URLs accessed we generated over one month contains 459452 requests, the total size of the objects is 3880998 Kbytes. Figure 4 shows the number of objects which are in set intervals of object size in the range up to 32 Kbytes, with the rightmost number indicating all objects in the range greater than 32 Kbytes.

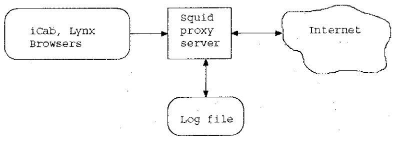

図 1 Creating the log file (step 1).

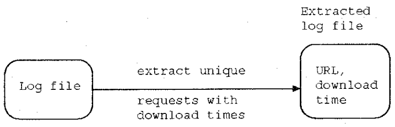

図 2 Creating the log file (step 2).

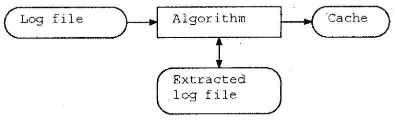

図 3 Creating the log file (step 3).

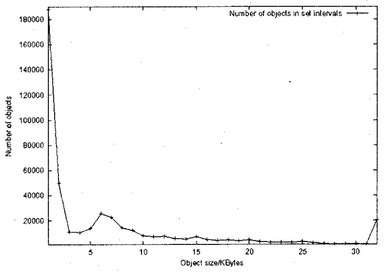

図 4 The numbers of downloaded objects in intervals of object size.

### 5.1 The whole Log File

In this experiment, we used the generated log file, and compare the results of our algorithms that cache according to response-delay time with the results for SIZE, LFU, LRU, and FIFO.

The results of this experiment show that the algorithm that caches according to response-delay time alone achieves the best result of any of the algorithms for total response-delay time (see Figure 5). We note that in Figure 5 the cache size affects the total response-delay time, that is, when the cache size is increased the total response-delay time falls.

We also conducted experiments on the hit and byte ratios. The response-delay-based caching algorithm achieved a very

(注1) : http://www.cs.berkeley.edu/logs/http/
(注2) : ftp://ftp.digital.com/pub/DEC/traces/proxy/

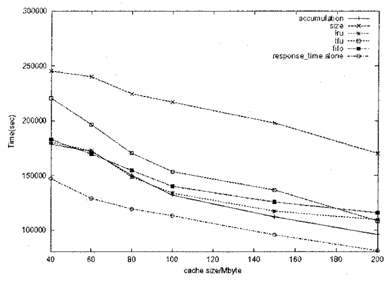

図 5 Comparing the total response-delay time for all algorithms on different cache sizes.

poor result for hit ratio, while the accumulative algorithm achieved both high high hit ratios and low response-delay times. The results for hit ratio are given in Figure 6.

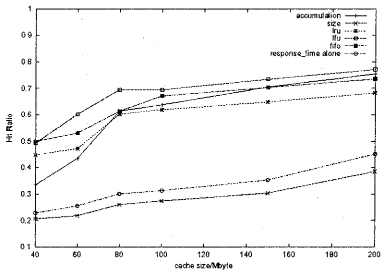

図 6 Comparing the hit ratio for all algorithms on different cache sizes.

Figure 7 shows the results for normalized response-delay time $T/RN$ with different cache sizes, where $T$ is the response-delay time, $R$ is the hit ratio, and $N$ is the total number of requests. Figure 8 is the inverse of Figure 7, that is, the results for inverse of the normalized response-delay time across cache size.

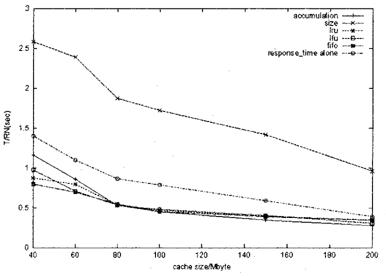

図 7 The normalized response-delay time for all algorithms using the whole file.

## 5.2 Objects between 0 and 4 Kbytes

In this experiment, we extracted the log entries for objects of size between 0 and 4 Kbytes to a separate file, and

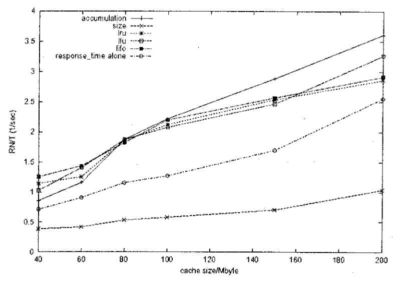

図 8 The inverse normalized response-delay time for all algorithms using the whole file.

then carried out the same experiments on this file as were done for all of the data in the previous section, 5.1. Figure 9 shows the total response-delay time for each of the algorithms, this result is the same whether the cache size is equal to or greater than 60 Mbytes, because of the smallness of the objects. When cache size is 40 Mbytes, however, the algorithm that caches only according to the response-delay time still achieves the best results. The results for hit ratio are

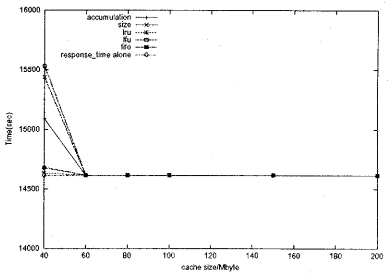

図 9 Comparing the total response-delay times for objects with sizes between 0 and 4 Kbytes.

shown in Figure 10.

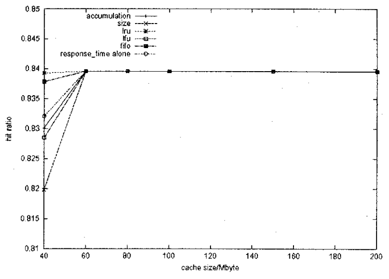

図 10 Comparing the hit ratios for objects with sizes between 0 and 4 Kbytes.

The normalized response-delay times $T/RN$ for caches with various sizes are shown in Figure 11.

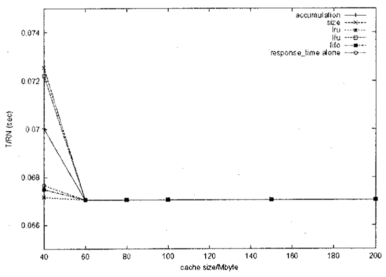

図 11　The normalized response-delay times for objects with sizes between 0 and 4 Kbytes.

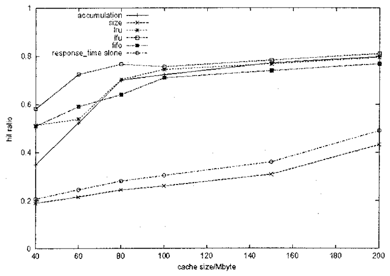

図 13　Comparing the hit ratio using large objects for all algorithms on different cache sizes.

## 5.3 Objects larger than 4 Kbytes

We extracted the log entries for objects that are larger than 4 Kbytes to a separate file, and then carried out the same experiments on this file as were done for all of the data in Section 5.1. With any cache size, our response-delay-based caching algorithm achieves the best result for response-delay time in this case, with the accumulative algorithm second-best, as is shown in Figure 12. The results for hit ratio are shown in Figure 13, where the accumulative algorithm achieves a high hit ratio. The results for normalized response-delay times $T/RN$ for different cache sizes are shown in Figure 14, and Figure 15 is the graph for inverse time.

In Figure 12, we see that when the cache size increases the total response-delay time decreases. In Figure 14, we see that the normalized response-delay time decreases as the cache size increases. Figure 14 shows that the normalized response-delay time of the accumulation algorithm is best when the cache size is equal to or greater than 80 Mbytes. The accumulative algorithm is better than the LFU and LRU by 18% and 20%, respectively, when the cache size is 200 Mbytes.

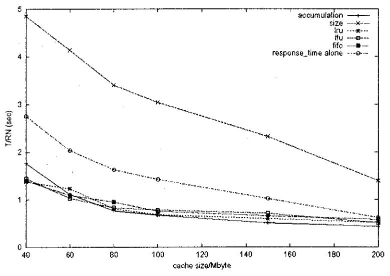

図 14　The normalized response-delay time for large objects.

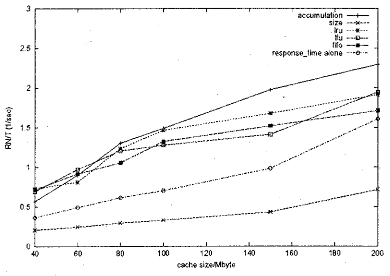

図 15　The inverse normalized response-delay time using large objects.

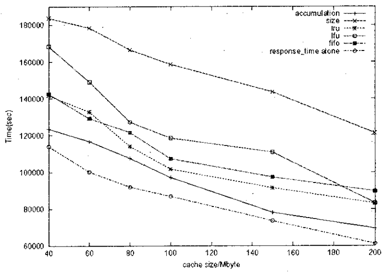

図 12　Comparing the total response-delay times for large objects with all algorithms and across various cache sizes.

## 6. Discussion and Analysis

Consider the following symbols:

- $T$: total response-delay time
- $C$: cache size
- $V$: available cache size
- $m$: number of objects in the cache
- $A$: a set of the objects in the cache
- $M$: number of requested objects
- $N$: number of all requests
- $O_i$: object $i$
- $n_i$: number of requests for object $i$
- $t_i$: download time for object $i$

- $S_i$: size of object $i$
- $R$: hit ratio
- let $O_1, O_2, \ldots, O_m$ be objects in the cache

Since the total size of the objects in the cache is less than or equal to the cache size, we can write:

$$\sum_{i=1}^{m} S_i \leqq C \tag{5}$$

Let the initial values for $A \leftarrow \{\phi\}$, $V \leftarrow C$ and $m \leftarrow 0$, every time a new object $O_j$, where $m+1 \leqq j \leqq M$, is placed in the cache, the following equations hold:

$$V \leftarrow V - S_j \tag{6}$$

$$m \leftarrow m+1, \quad V \geqq S_j \tag{7}$$

$$A \leftarrow A + \{O_j\} \tag{8}$$

When an existing object $O_i$ leaves the cache, the following equations hold:

$$V \leftarrow V + S_i \tag{9}$$

$$m \leftarrow m-1, \quad V < S_j \tag{10}$$

$$A \leftarrow A - \{O_m\} \tag{11}$$

The objects in the cache for the basic algorithm $A_{basic}$ are sorted such that:

$$t_i \geqq t_{i+1}, \quad 1 \leqq i \leqq m-1 \tag{12}$$

When a new object $O_j$ is requested, if $V < S_j$ and $t_j > t_m$ then $O_m$ leaves the cache until $V \geqq S_j$, at that time $O_j$ is placed in the cache.

While the objects in the cache for accumulative algorithm $A_{acc}$ are sorted such that:

$$t_i n_i \geqq t_{i+1} n_{i+1}, \quad 1 \leqq i \leqq m-1 \tag{13}$$

When a new object $O_j$ is requested, if $V < S_j$ and $t_j n_i > t_m n_m$ then $O_m$ leaves the cache until $V \geqq S_j$, at that time $O_j$ is placed in the cache.

We can see that the set $A_{basic}$ of objects in the cache for the basic algorithm differs from the set $A_{acc}$ for the accumulative algorithm from equations (12) and (13), respectively, which leads to the difference in the total response delay times and the hit ratio for both algorithms.

The following formula represent the total response delay times for objects not placed in the cache, i,e., they must be fetched each time they are requested:

$$T = \sum_{i=1}^{M} t_i, \quad O_i \notin cache$$

An object $O_i$ with large $t_i$ is placed in the cache and when requested again its fetching time is not added to $T$, so $T$ is a summation of small $t_i$ which leads to small $T$.

Since users do not like to wait for slow objects, then objects with large $t_i$ are not requested so many times and they are not almost served from the cache. Recall the computation of the hit ratio from equation (3), if $n_i$ is small, then the hit ratio is low. If objects with large $n_i$ are placed in the cache then the hit ratio is increased.

For the basic algorithm, objects with long $t_i$ and small $n_i$ are placed in the cache, that achieves low total response delay times and low hit ratio. While for accumulative algorithm, objects with long $t_i$ and large $n_i$ are placed in the cache because of the consideration of $t_i n_i$, which achieves low total response delay times and high hit ratio.

The following equations compute the total response-delay times for all requested objects, when the basic and accumulative algorithms, respectively:

$$T_{basic} = \sum_{i=m_{basic}+1}^{M} n_i t_i + \sum_{i=1}^{m_{basic}} t_i + \alpha_1 \tag{14}$$

where $t_1 \geqq t_2 \geqq \ldots \geqq t_{m_{basic}} \geqq t_{m_{basic}+1} \geqq \ldots \geqq t_M$.

$$T_{acc} = \sum_{i=m_{acc}+1}^{M} n_i t_i + \sum_{i=1}^{m_{acc}} t_i + \alpha_2 \tag{15}$$

where $n_1 t_1 \geqq n_2 t_2 \geqq \ldots \geqq n_{m_{acc}} t_{m_{acc}} \geqq n_{m_{acc}+1} t_{m_{acc}+1} \geqq \ldots \geqq n_M t_M$, and $\alpha_1$, $\alpha_2$ are the initial download time deviation.

## 7. Conclusion

In this study, we have taken the time that users have to wait to receive the data they request into account in a scheme for caching where objects are replaced on the basis of object response-delay time to shorten the user waits as short as possible. Experimental results show that the response-delay-based caching algorithm is the best algorithm for reducing the waiting times, that is, it shortens the times clients have to wait to receive the data they request by as much as possible. The factor we consider in the accumulative algorithm is the multiple of the number of requests and the response-delay times for the respective objects, and the results show that using this multiple improves on the efficiency of our basic algorithm. The accumulative algorithm which we introduced increases the hit and byte ratios above the values for the basic algorithm and better results than any algorithm other than the basic algorithm for total response-delay time.

We intend to look for another factor or factors that can achieve better results further, shortening response-delay time in addition to achieving higher hit and byte ratios. We need further experiments and study to prove the effect of the object size on the total response-delay time and find the corresponding relation. We are also going to analyze the total response-delay times $T_{basic}$ and $T_{acc}$ for Equations (14), (15), respectively.

## Acknowledgment

We would like to thank Professor Yasuhiro Ohtaki and Shusuke Okamoto of Ibaraki University for their opinions, and for the ideas and help they gave using this study.

## 文　献

[1] A. Chankhunthod and P. B. Danzing, "A Hierarchal Internet Object Cache," *Harvest Cache Project,* Nov. 1995.
[2] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, "Removal Policies in Network Cache for World-Wide Web Documents," *Computer Communication Review, ACM SIGCOMM,* vol.26, No.4, Oct. 1996.
[3] A. Luotonen, Web Proxy Server, *Prentice-Hall,* 1998.
[4] P. Cao, S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. of the USENIX Symposium on Internet Technologies and Systems,* Dec. 1997.
[5] D. Wessels, *Web Caching,* O'Reilly, 2001.
[6] A. Loutonen and K. Altis, "World-wide web proxies," *Preliminary Proc. of the First Int'l World Wide Web Conf.,* Apr. 1994.
[7] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, E. A. Fox, "Caching Proxies: Limitations and Potentials" at http://ei.cs.vt.edu/ succeed/www4/www4.html
[8] "Squid: A User's Guide" at http://squid-docs.sourceforge.net/latest/html/c23.htm
[9] R. P. Wooster, Marc Abrams, "Proxy Caching That Estimates Page-Load Delays," *Proc. of 6th Int'l World Wide Web Conf.,* April, 1997.
[10] http://www.squid-cache.org/Versions/v2/2.4/
[11] http://squid-docs.sourceforge.net/latest/html/
[12] D. A. Menasce, V. A. F. Almeida, *Capacity Planning for Web Performance, Metrics, Models, and Methods,* Prentice Hall, 1998.
[13] D. E. Commer, *Internetworking with TCP/IP,* Prentice-Hall, 2000.