WEB DELAY ANALYSIS AND REDUCTION BY USE OF LOAD BALANCING OF A DISPATCHER-BASED WEB SERVER CLUSTER

Ying-Wen Bai and Yi-Chao Wu
Department of Electronic Engineering
Fu Jen Catholic University
Taipei, Taiwan, 242, R.O.C.
bai@ee.fju.edu.tw

ABSTRACT
The loading of web servers can be very heavy and a delay in accessing certain Web sites can be substantial. Hence, the reduction of web delay is one of the important topics of research for improving the quality of service of Web pages. Thus, in this paper, we propose the web delay analysis and the web delay reduction by using load balancing coupled with the dispatcher-based approach. First, we propose a queueing model to represent the web delay. Second, we propose a mathematical model to obtain the least possible web delay by the use of load balancing method. By using the first and second portions as simulated by QNAT, we find the cluster of web servers may not have incorporated any load balancing for all situations. Thus, we propose two strategies to justify the existence of load balancing. The first one is a simulation module that we design by using Virtual C++. The second is a Hessian matrix improve the computation efficiency of web delay. In addition, to simplify our analysis, we produce a pre-compute table in order that people may judge the load balancing of web servers without using complicated computation. Overall, we learn that load balancing will reduce the web delay substantially. Moreover, by meeting the specific conditions of the load balancing scheme, we can have a potential to obtain a web delay reduction of 38.46%.

KEY WORDS
Web Server, Load Balancing, Queueing Model, Simulation Module, Hessian Matrix.

1. INTRODUCTION
Since the information traffic on the World Wide Web increase at the rate of about 2 ~3 times per year and the average login-time of a user on a Web page is much longer than before, the “web delay” or the system “response time” is one of the main factors of the service quality of a Web page. In addition, the network bandwidth and the connection strategy can also affect the total latency sources. Many ways such as “Caching architectures”, “Prefetching” and “Content distribution network”, and “load balancing” have been studied as potential techniques for reducing the web delay [1-4].

Due to the stochastic characteristics of web server operation and Internet transmission, web delay can be very random. Therefore it can be difficult to estimate the delay precisely. Our research in this paper includes the finding of the key factors utilized in estimating web delay and using one of these load balancing approaches to reduce web delay [2]. Based on the load balancing of a dispatcher-based approach, we provide a mathematical model to investigate the web delay of a cluster of web servers and to find the dependent factors [1-5].

First, we choose a dispatcher-based web server cluster because it is commonly used and has not been studied in any previous quantified analysis. For that web server cluster, we design a queueing model to conduct a quantification analysis and find the mathematical relation to the web delay of the various dispatch-factors. Based on this result, one can adjust the system operation parameters to obtain the least possible web delay.

Second, we use the mathematical software package QNAT [6] to conduct the simulation a. In addition, we propose a mathematical model based on a queueing model to compute either the web delay or the system response time E(t). The mathematical model will let us know the property of the web delay E(t). Then, we use the QNAT and TK Solver software packages to verify the correctness of our mathematical model. Furthermore, we design a new simulation tool to reduce the simulation time for finding the load balancing condition of a cluster of web servers. We also use a modified version of Hessian Matrix called H [E(t)] to show how we can locate the operation region of the load balancing of a cluster of web servers. Finally, we provide a pre-compute table to know how to find load balancing without using excessively complicated computation [7,8].

The rest of this paper is organized as follows. In Section 2, the main factors of web delay are described. In Section 3, a couple of load balancing schemes are present
and reasons are given why we choose the dispatcher-based method. In Section 4, the queueing model is described. In Section 5, QNAT simulation results on the load balancing of a cluster of web servers is described. In Sections 6 and 7, the mathematical model is described and verified. In Section 8, our simulation module for finding the required load balancing is described. In Section 9, the Hessian matrix is used to analyze web delay. Finally, in Section 10, some conclusions are drawn from our research.

2. The Main Factors of Web Delay
2.1 Latency Sources of System

When we use a Web browser or open a version of Windows, a series of processes of computer systems and networks will be processed in the background. Every process has a random delay time of about millisecond to second. For example, in a client-server system, one sends a request to the browser in the web site, and then the browser will service the accepted request. Most of the web contents must wait for other module responses such as software or hardware etc... Although in the best of situations, a basic amount of time is required to execute their function no matter whether by link or device services. So “latency sources” can be the cause of the most basic delay and can make it difficult to improve our common computer networks [2].

3. The Previous Proposed Approaches
3.1 Client-based Approach

Client-based approach routes the request to one of the nodes on the web clusters to a server based on software of the client-site. It is divided into web client and client-side proxies. The state management of the nodes is very difficult when network traffic increases. Therefore, as the client-based approach is not often used, we won’t consider using it [1, 2].

3.2 DNS-based Approach

Due to the increase of web traffic and longer URL names, we hope to make a simple virtual interface to communicate with others. This interface could make users not to know how to do inside transparently. A DNS-based approach just transforms the site name of the distributed Web system’s nodes to IP address. So, we must design a couple of solutions in order to distribute all client requests to the most suitable server. Of course, many intermediate name servers can cache logical-to-IP-address mapping to reduce the amount of network traffic between a client and the cluster DNS. Therefore DNS-based TTL (Time-To-Live) can be used to determine how much time will be required to find an intermediate name server. Otherwise the DNS cluster will select a suitable web server. This approach is a good idea, but the cluster DNS server often becomes a bottleneck due to too much traffic passing through intermediate name server [1, 2, 4, 7].

3.3 Dispatcher-based Approach

In order to centralize the request scheduling and control client-request routing completely, we have designed a dispatcher-based approach. The routing request among servers is transparent, but it is also the same when dealing with addresses in the URL level on DNS. A typical dispatcher has its own simple virtual IP address (IP-SVA). The dispatcher will define its own personal address based on its server and its different protocol level based distinct structures such as packet rewriting, packet forwarding, or HTTP redirection. The dispatcher will select an algorithm to choose a web server to manage incoming requests to minimize processing delay as shown in Fig. 1 [1, 4].

3.4 Server-based Approach

The server-based approach uses a two-level system of dispatch. It distributes the client requests on the DNS of the web system to the Web-server nodes first. Then every server distributes the request to other system servers again. Of course, its advantage is to conduct load balancing more effectively. In opposition, implements and managements are more inconvenient [1, 4].

Based on easily implementation and management, and the greatest effect of load balancing, we determine to use the dispatcher-based approach, because we just only manage the dispatcher with respect to the management of the dispatcher-based. The initial dispatcher architecture is shown Fig. 2. Based on this architecture, we will propose a queuing model to easily analyze the system response time or other performance index.
4. Queueing Model

Our architecture adopts one of dispatcher-based approaches. First, this dispatcher distributes the requests to server1 with the ratio “a” or server2 with the ratio “1-a”. The unfinished requests are automatically sent back to the dispatcher and are distributed again. The finished requests are sent to the output. We hope to have the system response time, $E(t)$ when we distribute the ratio “a”, “b”, and “c” suitably. In our dispatcher system as shown Fig. 2, server1 and server2 only have one server to serve. Our packet is never discarded; therefore the buffer is assumed infinite. In our design, on every server the service rate must be larger than arrival rate.

5. Simulation Result by QNAT

In order to learn more about the delay time of the balance queue in the corresponding model as shown Fig. 3 as the software package QNAT has many advantages to analyze the characteristics of the queueing model such as throughput, utilization and response time, we choose QNAT as a simulation tool. In our simulation, we choose the system response time, $E(t)$ as major performance index., and $E(t)$ can also serve as the major part of the web delay. Before we use QNAT, we need choose some fixed factors and other controlled factors. The fixed factors are $\lambda$, $\mu_1$, $\mu_2$, and $\mu_3$. The controlled factors are $\mu_1$, $\mu_2$, $\mu_3$. The dispatch ratio “a” is the factor we can change. Due to the covering simulation of the whole system characteristics, the dispatch ratio “a” range is from 0.1 to 0.9 by step size of 0.1 every time.

![Fig. 2 Basic Dispatcher-based Approach architecture](image)

![Fig. 3 Queueing model for basic Dispatcher-based Approach architecture](image)

![Fig. 4 Curve representations for simulation results](image)

Note: $m_1: \mu_2 = 4, \mu_3 = 10$ ; $m_2: \mu_2 = 5, \mu_3 = 12.5$ ; $m_3: \mu_2 = 6, \mu_3 = 15$
Table 1 A Comparison of the system response time using different service rates and dispatch ratios

<table>
<thead>
<tr>
<th>a</th>
<th>1-a</th>
<th>$E(t)$ by QNAT</th>
<th>$E(t)$ by QNAT</th>
<th>$E(t)$ by QNAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.9</td>
<td>1.70536</td>
<td>1.38724</td>
<td>1.26925</td>
</tr>
<tr>
<td>0.2</td>
<td>0.8</td>
<td>1.61111</td>
<td>1.37994</td>
<td>1.27697</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>1.69318</td>
<td>1.43367</td>
<td>1.31562</td>
</tr>
<tr>
<td>0.4</td>
<td>0.6</td>
<td>2.23077</td>
<td>1.60029</td>
<td>1.40338</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>None*</td>
<td>None*</td>
<td>None*</td>
</tr>
<tr>
<td>0.6</td>
<td>0.4</td>
<td>None*</td>
<td>7.08620</td>
<td>2.0679</td>
</tr>
<tr>
<td>0.7</td>
<td>0.3</td>
<td>None*</td>
<td>None*</td>
<td>4.54867</td>
</tr>
<tr>
<td>0.8</td>
<td>0.2</td>
<td>None*</td>
<td>None*</td>
<td>None*</td>
</tr>
<tr>
<td>0.9</td>
<td>0.1</td>
<td>None*</td>
<td>None*</td>
<td>None*</td>
</tr>
</tbody>
</table>

None* : The system response time is negative or infinite

From Table 1, we learn that the results with load balancing are better than without load balancing by about $38.46\% = \left[ \frac{(2.23077-1.61111)}{1.61111} \right] \times 100\%$, which from the first column of the Table 1 when $\mu_2 = 4$, $\mu_3 = 10$

6. Mathematical Model for the Load Balancing

In order to analyze effectively, we need to write down the system response time, $E(t)$, whose functions include $\lambda$, $\mu_1$, $\mu_2$, $\mu_3$, a, b, and c based on the analysis of a typical queueing model such as Fig. 3 with a consideration of four portions.

First, the service rate $\mu$ and arrival rate $\lambda$ are the most basic factors of the queueing model in Fig. 3. The arrival rate of the dispatcher server, the arrival rates of server1 and server2 are not the same because we use the dispatcher-based approach. Therefore we must assign the arrival rate of the dispatcher server, and the arrival rates of server1 and server2. We have indicated below is the arrival rate we assigned. Where $\lambda$ represents the initial arrival rate of the whole system, $\lambda_1$ represents the equivalent arrival rate of the dispatcher, $\lambda_2$ represents the equivalent arrival rate of the server1 and $\lambda_3$ represents the equivalent arrival rate of the server2. The “a” represents the dispatch ratio from the dispatcher to server1. The “b” represents the unfinished work feedback ratio from server1 to the dispatcher. The “c” represents the unfinished work feedback ratio from server2 to the dispatcher.

$$\lambda_1 = \frac{\lambda}{1-a}$$  \hspace{1cm} (1)
$$\lambda_2 = \frac{a\lambda}{1-ab-c+ac}$$  \hspace{1cm} (2)
$$\lambda_3 = \frac{(1-a)\lambda}{1-ab-c+ac}$$  \hspace{1cm} (3)

Second, we will compute the utilization, or capacity $\rho$. Usually, the $\rho$ is a key factor on the queuing model because it shows how much the capacity is in every server buffer. In our analysis, we will let $\rho$ range from 0 to 1. If $\rho$ is greater than 1, we must deal with the overflow analysis in queuing theory. Then $\rho_1$, $\rho_2$ and $\rho_3$ represent the dispatcher capacity, the server1 capacity, and the server2 capacity respectively.

$$\rho_1 = \frac{\lambda_1}{\mu_1} = \frac{\lambda}{(1-ab+ac-c)\mu_1}$$ \hspace{1cm} (4)
$$\rho_2 = \frac{\lambda_2}{\mu_2} = \frac{a\lambda}{(1-ab+ac-c)\mu_2}$$ \hspace{1cm} (5)
$$\rho_3 = \frac{\lambda_3}{\mu_3} = \frac{(1-a)\lambda}{(1-ab+ac-c)\mu_3}$$ \hspace{1cm} (6)

Third, the average number of packets in a queue:

$$E(k_n) = \frac{\rho_n}{1-\rho_n}, n = 1,2,3$$ \hspace{1cm} (7)

Fourth, the average time for a single packet to get through the queue:

$$E(t) = \sum_{n=1}^{3} \frac{E(k_n)}{\lambda}, n = 1,2,3$$ \hspace{1cm} (8)

So we can find the $E(t)$ function as follows:

$$E(t) = \frac{1}{(1-ab+ac-c)\mu_1 - \lambda} + \frac{a}{(1-ab+ac-c)\mu_2 - a\lambda} + \frac{1-a}{(1-ab+ac-c)\mu_3 - (1-a)\lambda}$$ \hspace{1cm} (9)

From Eq. (9), we learn $E(t)$ as the variable relations among $\lambda$, $\mu_1$, $\mu_2$, $\mu_3$, a, b and c. Next, we use the software package “TK Solver” to verify that our mathematical model of the specific situation is correct.

7. Verification of Our Mathematical Model

Because our mathematical model includes the feedback mechanism, we design a program or use the relative mathematical software to verify the model’s effectiveness and exactness. Therefore, we choose TK Solver to verify our mathematical model. First we must point out that TK Solver is not only the mathematical tool but also the queue simulation tool. The reason we use TK...
Solver is that it is unlike the general programs that declare the variables and write the statement. It only inputs parameters for the simulation of the E(t) function, and one just has to define the input variables and the output variables. The results from our use of TK Solver and QNAT. We obtain these very approximate results by using TK Solver and QNAT based on successive simulation. Hence, we are sure the E(t) function we have computed is correct. We also simulate a non-ideal distribution. We find that the results are only slightly different, but these results don’t influence the existence of concave.

8. The Simulation Module

Although the computation of the E(t) is correct, we still need to recompute a system response time by using TK Solver or QNAT for every specific set of the parameters. The computation is not effective, especially when we want to simulate a lot of system response time, the E(t), with respect to a wide range of parameters. It can be a very difficult problem because the computation cost will be expensive. Although on our simulation we regulate the dispatch ratio “a” suitably, it is not able to have load balancing every time. Thus, we can’t use QNAT or TK Solver to simulate from a=0.1 to 0.9 by adding “a” to 0.1 every time. It can be a very complex problem to locate the load balancing of the model. If one is lucky, one may find the load balancing for the dispatch ratio “a” from 0.1 to 0.9. But if one is unlucky, one may not be to find the load balancing and will have to perform a simulation ranging from 100 to 1000 times. Because of this reason, we need to design a simulation module which can locate the load balancing. It can let one input the fixed factors and the controlled factors, and it can indicate when there is load balancing. If the module finds the load balancing, it will record what the λ, μ2, and μ3 are. Hence, one need not input the variable parameters again by using TK Solver or QNAT simulation. Currently, we design the simulation module just only for our model. The flowchart of the simulation module is shown in Fig. 5.

Table 2 shows the simulation results by using our simulation module and the QNAT. We can find a very limited error between two methods, but it is minor.

<table>
<thead>
<tr>
<th>a</th>
<th>E(t) by our simulation module</th>
<th>E(t) by QNAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1.70536</td>
<td>1.70536</td>
</tr>
<tr>
<td>0.2</td>
<td>1.61111</td>
<td>1.61111</td>
</tr>
<tr>
<td>0.3</td>
<td>1.69318</td>
<td>1.69318</td>
</tr>
<tr>
<td>0.4</td>
<td>2.23077</td>
<td>2.23077</td>
</tr>
</tbody>
</table>

9. Using the Hessian Matrix

Though our simulation module, one can quickly determine if the system has load balancing. However, the decision of the load balancing still necessitates complicated computation. We hope to be able to predict the load balancing more efficiently. The following description shows how to use the Hessian matrix to build the pre-computation table for the determining the load balancing.

The first step: find the differentiation of the Eq. (9), E(t), with respect to “a”, “λ”, “μ2”, and “μ3”. The second step: find the differentiation of Eq. (10) with respect to “a”, “λ”, “μ2”, and “μ3”. The third step: find the differentiation of Eq. (11) with respect to “a”, “λ”, “μ2”, and “μ3”. The fourth step: find the differentiation of Eq. (12) with respect to “a”, “λ”, “μ2”, and “μ3”. The fifth step: find the differentiation of Eq. (13) with respect to “a”, “λ”, “μ2”, and “μ3”. The sixth step: fill in the Hessian matrix of E(t): H[E(t)]

$$H[E(t)]=
\begin{bmatrix}
\frac{\partial E(t)}{\partial a} & \frac{\partial E(t)}{\partial \mu_2} & \frac{\partial E(t)}{\partial \mu_3} & \frac{\partial E(t)}{\partial \lambda} \\
\frac{\partial E(t)}{\partial \mu_2} & \frac{\partial E(t)}{\partial \mu_2} & \frac{\partial E(t)}{\partial \mu_3} & \frac{\partial E(t)}{\partial \mu_2} \\
\frac{\partial E(t)}{\partial \mu_3} & \frac{\partial E(t)}{\partial \mu_3} & \frac{\partial E(t)}{\partial \mu_3} & \frac{\partial E(t)}{\partial \mu_3} \\
\frac{\partial E(t)}{\partial \lambda} & \frac{\partial E(t)}{\partial \lambda} & \frac{\partial E(t)}{\partial \lambda} & \frac{\partial E(t)}{\partial \lambda}
\end{bmatrix}$$

Fig. 5 The flowchart of the simulation module for finding the load balancing
In this paper, we propose four results. First, the basic dispatcher-based approach architecture may not have load balancing in any situation. Second, we can compute the value of the system response time, $E(t)$, based on the queuing model of our dispatcher-based approach architecture and the mathematical model we design without using complicated computation. Third, based on our simulation module, we can determine how to distribute the tasks among three computers in our dispatcher-based architecture to have load balancing. Overall, our simulation module can improve the whole simulation effectiveness. The fourth result of our paper is the use of Hessian matrix to obtain the relation or a table so that we can locate the load balancing. This pre-compute table provides a criterion to judge the existence of load balancing.

REFERENCES